ISSN 0316-6295

A FIRST-ORDER DYNAMIC LOGIC FOR PLANNING

Henry A. Kautz

Technical Report CSRG-144

May 1982

COMPUTER SYSTEMS RESEARCH GROUP
UNIVERSITY OF TORONTO

A FIRST-ORDER DYNAMIC LOGIC FOR PLANNING

Henry A. Kautz

Technical Report CSRG-144

May 1982

The Computer Systems Research Group (CSRG) is an interdisciplinary group formed to conduct research and development relevant to computer systems and their application. It is jointly administered by the Department of Electrical Engineering and the Department of Computer Science of the University of Toronto, and is supported in part by the National Research Council of Canada.

A FIRST-ORDER DYNAMIC LOGIC FOR PLANNING

Henry A. Kautz

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada

A thesis submitted to the
School of Graduate Studies
University of Toronto
in partial fulfillment of the requirements
for the Degree of Master of Science

Abstract

This thesis explores the statement and synthesis of robot plans in dynamic logic. Stanley Rosenschein [Rosenschein 81] has developed a formal framework in propositional dynamic logic; I extend his work to a first-order language. The approach offers a "possible worlds" semantics for plans, and a natural treatment of disjunctive and quantified goals and hierarchical plans. Dynamic logic may bridge the gap between earlier formal treatents of planning in the situational calculus, and the more popular and efficient state-space planning systems such as STRIPS.

Acknowledgements

I would like to thank Professor Ray Perrault for his supervision of my work, and my second reader, Professor John Mylopoulos. Support was provided by the University of Toronto.

Contents

CHAPTER 1

INTRODUCTION

This thesis explores a new formulation of the robot planning problem in terms of dynamic logic (DL). The general form of a planning problem is to find a sequence of actions which will transform a set of initial world states into a set of goal states, where a set of states is usually described by a logical formula. Early researchers reduced planning to theorem proving in a first-order language called the situational calculus [McCarthy and Hayes 69]. This approach offers a well-defined semantics for plans (namely, any "standard" semantics of first-order logic), a rich vocabulary for describing world states, and a provably correct and complete planning process. But the situational calculus has nothing to say about how the search for plans is structured; without such control knowledge, a planner is hopelessly inefficient.

The STRIPS series of planners [Fikes and Nilsson 71] concentrated on such efficiency issues, but neglected many of the positive features of the "purely logical" approach. World descriptions were limited to conjunctions of literals, making disjunctive and quantified knowledge inexpressible; actions were defined by simple syntactic operators ("add lists" and "delete lists") which lacked clear semantics.

Stanley Rosenschein has formulated planning in propositional dynamic logic, a modal logic developed for program verification [Harel 79] [Pratt 76], and has given a complete "bigression" algorithm ("bidirectional progression and regression") for synthesizing a broad class of plans [Rosenschein 81]. As in the situational calculus, actions are described by logical axioms, leading to provably correct plans; as in STRIPS, an explicit algorithm structures the search. The framework naturally and elegantly handles conditional actions and hierarchical plans.

I will define a first-order dynamic logic suitable for stating planning problems, and extend Rosenschein's algorithm to handle a subset of the language. Much of the work centers on calculating the effect of a possibly non-deterministic action upon our knowledge of the world when the simplifying STRIPS assumptions are abandoned.

Many problems which are difficult to even state using STRIPS-like operators can be readily solved by a DL-planner. The familiar blocks-world repertoire is immediately expanded to include disjunctive goals, as in the command, "Build a red tower OR a blue tower." Deduced information is no longer problematical. Given axioms to describe the predicate NEXTTO in terms of ON, no special rules are needed to "delete" NEXTTO(BLK1, BLK2) from the world description when ON(BLK1, BOX3) or ON(BLK2, BOX3) becomes false. Bound variables may appear in goal or action descriptions, as in "Put SOME red block in the box, and keep ALL the others in place", or "The action IRRADIATE(x) sterilizes ALL the fruit-flies in dish x."

Consider the "3-socks" problem. You stand in a lightless room before a dresser full of loose socks, some black and some white. How many socks must you pull from the dresser before you can be sure that you have a matching pair? Not only is the goal indefinite -- get two black socks OR two white socks, and I don't care which socks in particular -- but the "pull out a sock" action is non-deterministic. Yet the problem is solvable because its non-determinism is constrained in a known way: the sock will be distinct from those obtained earlier and will be either black or white.

DL-action axioms can encode more information about "pull out a sock" than can any simple add-list/delete-list pair. It is unlikely, of course, that any finite representation will succeed in capturing all conceivably relevant effects of a real-world action. It is important, therefore, that the representation be monotonically extensible. Every DL-axiom specifies some partial effects of an action; unless the system is made inconsistent, adding more axioms will never invalidate an old plan. The opposite is generally true of STRIPS; one is forced to to add new actions rather than redefining old ones if compatibility is to be retained.

The price the DL approach pays for its flexibility is the need for "frame axioms", axioms which specify all the predicates which a action does NOT affect. But frame actions are useful for describing actions with possible side-effects.

This thesis shall not concentrate on particular heuristics for choosing "good" actions when building a plan. Instead, two classic

A.I. techniques for reducing the size of the search space shall be applied to the DL framework. The first involves merging branches of the search tree by allowing incompletely-specified actions in partial plans [Sussman 75].

Perhaps the greatest efficiencies can be obtained through the use of plan hierarchies. This is commonly implemented (eg, ABSTRIPS [Sacerdoti 74] and NOAH [Sacerdoti 77]) by temporarily hiding "less important" details from the planning unit; when the details are filled in, the plan may be in error and must be patched up. The complexity of the patching process, unfortunately, weakens our faith in the ultimate correctness of the plan ad the overall efficiency of the strategy. The DL framework, on the other hand, suggests an exact hierarchy. The basic actions on one level are simply planning problems to be solved on lower levels. A subplan is guaranteed not to unexpectedly clash with other parts of the high-level plan.

I'll consider "good" decompositions of the blocks and "dungeons" worlds. One can specify how "general" a subplan should be, balancing the possibly higher cost of finding a very general plan against its greater potential for reuse.

Finally, I'll discuss some further extensions of the language and the algorithm. The system has not been implemented. The immediate concern is to demonstrate that the method is sound, and potentially better than anything currently available.

CHAPTER 2

INTELLECTUAL ORIGINS

In its broadest sense, "planning" incorporates what might be taken as the task of artifical intelligence:

> ..we want a computer program that decides what to do by inferring in a formal language that a certain strategy will achieve its assigned goal. [McCarthy and Hayes 69]

An intelligent program must possess an adequate representation of the world, including its own desires and abilities, and a mechanism for applying its knowledge to solve the problems at hand.

Much work has been done on robot planning systems. In the classic scenario, the system guides a robot in manipulating objects in a very simple model of the physical world. The problem presented to the system is to devise a sequence of actions, a plan, which will transform a given initial state into a goal state. The robot is the only active entity; its knowledge of the world is simply identified with the representation chosen for states and actions.

In recent years, exciting research has proceeded toward a more psychologically sophisticated theory of planning; for example, see [Cohen 78] [Moore 80] [Haas 81]. Prolegomena to the specification

of any planning system, however, are the questions:

--What formally is a plan , a planning problem, and a solution?

--How is the search for solutions organized? Is the strategy provably correct?

--For what class of problems is it complete?

Rosenschein's work provides an elegant set of answers which unify and clarify the approaches adopted by a number of earlier methodologies. This chapter briefly describes some of these influences and summarises (with very minor adjustments, to lead more smoothly into the extensions of chapter 3) the framework presented in [Rosenschein 81].

2.1 SITUATIONAL CALCULUS

A situation s is the complete state of the universe at an instant of time. A first-order logic which allows situations to be the values of terms is a situational calculus. [McCarthy and Hayes 69] provides a discussion of the philosophical underpinnings of the use of situations to describe action, causality, and ability. [Green 69a] reduced robot planning to theorem proving in a version of the situational calculus.

Each predicate takes an argument specifying a situation. For example, "on(A,B,S1)" asserts that (the entity denoted by) A is on B in situation S1. Some functions evaluate to actions. "Move(A,B,C)" is a term which stands for the action of moving object A from

location B onto C.

The function "result(a,s)" maps a state s and an action a to the state resulting from performing the action in s.

A planning problem is formulated as follows. The initial state is described by a wff all of whose predicates include some arbitrary situation S0. The desired final state is described by a wff whose common situational variable is existentially quantified. The problem is to find a constructive proof that the second wff follows from the first -- for example, that

on(A,B,S0) ⊃ Es on(A,C,s)

-- using a set of axioms which specify the results of the actions. An axiom for move might be

∀sxyz ((on(x,y,s) ∧ location(z)) ⊃
on(x,z, result(move(x,y,z), s)))

A constructive proof the problem wff actually finds the situation s such that on(A,C,s). In this case,

on(A,B,S0) ⊃ on(A,C, result(move(A,B,C),S0))

Thus, the plan to achieve on(A,C) given on(A,B) is to perform move(A,B,C).

Praiseworthy features of this approach include:

A solution to a solvable problem can be guaranteed to be found if a complete first-order proof strategy (such as resolution with answer-extraction [Green 69b]) is employed.

Correctness of plans is obvious, from correctness of the underlying proof procedure.

A well understood semantics exists (namely, any typical semantics for first-order logic) for plans and planning problems.

There is extreme flexibility in stating axioms and planning problems; there are no inherent objections to including quantifiers, disjunctions, additional functions, etc.

Yet the situational calculus says nothing about how the search for solutions is structured. Without adequate control knowledge, any theorem prover will be excessively slow.

Axioms must specify what each action does not change. In order to derive

$$(on(A,B,S0) \wedge location(C) \wedge on(D,E,S0)) \supset (on(A,C, result(move(A,B,C),S0)) \wedge on(D,E, result(move(A,B,C),S0)))$$

an axiom is required to the effect that an object not being moved stays in the same place:

$$\forall svwxyz\ ((on(v,w,s) \wedge w \neq x) \supset on(v,w, result(move(x,y,z),s)))$$

The computational problem of dealing with a large number of such axioms is called the frame problem.

Finally, the notation is rather awkward. One may also feel that the treatment of situations, actions, and objects -- all merely elements of the domain of interpretation -- is overly homogeneous.

2.2 STATE SPACE MODELS

A family of situations can be modeled by a logical formula, its state description. E.g., on(A,B) represents all situations where (the entity denoted by) A rests on B. Planning can be performed by searching the space of state descriptions, for a path from the initial state description to one which entails the goal description. Actions are modeled by operators which change one state description into another.

The STRIPS series of planners [Fikes and Nilsson 71] avoided the frame problem by limiting state descriptions to sets (conjunctions) of literals, and representing actions by syntactic operators which transformed descriptions by explicitly adding and deleting propositions. For instance, the move operator can be written

```
move(x,y,z)
    precondition list:  on(x,y)
    add list:  on(y,z)
    delete list:  on(x,y)
```

The instantiation move(A,B,C) of move can only be applied to state descriptions containing on(A,B), and its effect is to replace that proposition with on(B,C).

STRIPS decouples search from theorem proving. Theorem proving is employed within a state description to discover what operators are applicable, and what goals have been achieved. Search is directed by a means-end analysis algorithm [Newell and Simon 63]. An operator is chosen whose effects (add list) reduce the difference between the goal and the current state description. If the operator is applicable to the current state, it is immediately applied;

otherwise, its precondition becomes a new goal to be achieved. This "backward-chaining" search is goal-oriented, but state descriptions are always developed in a forward direction.

STRIPS' handling of conjunctive and disjunctive goals is fairly weak. It tries to achieve each literal in a conjunctive goal in a linear order; if an already-achieved goal is undone, planning is reattempted with the literals in a different order. STRIPS always attempt to achieve one disjunct of a disjunctive goal.

STRIPS made great strides in efficiency. But the non-logical nature of its operators make it difficult to formally prove the correctness or completeness of the system; indeed, its strategy is demonstrably not complete. The frame conditions for actions with possible side effects, so-called influential operators [Waldinger 77], can be too complex to be represented by the STRIPS default rule. Nor is it always reasonable to assume that the machine's knowledge of the world can always be represented by a conjunction of literals.

A problem solver can also search by transforming the goal description, in stages, to the initial state description, by applying the inverses of the action operators. Passing a relationship back through an operator is called regression. An uninformed forward or backward breadth-first search can be the basis for a complete planning strategy [Nilsson 80].

The RSTRIPS system [Waldinger 77] improved STRIPS' handling of conflicting conjunctive goals. When an attempt to achieve a particular goal literal interferes with an already achieved goal,

the offending goal literal is regressed back through the partially-constructed plan, in the hope that it will be easier to achieve at an earlier point in the plan. Its coverage of the solution space, however, is still incomplete and not completely characterized.

The REFLECT planner [Dawson and Siklossy 77] did perform pure backward search, regressing entire goal descriptions, without focusing on achieving particular goal literals. Simple STRIPS operators and pruning heuristics (similar in spirit to those that appear in Rosenschein's algorithm) allow REFLECT to solve problems requiring fairly long solutions.

2.3 HIERARCHICAL PLANNING

The space searched by a general planning system tends to grow exponentially as the distance between the initial and goal states increases. Various approaches reduce search by asserting a "vertical" control structure.

The creators of STRIPS experimented with using generalized versions of previously solved plans as "macro operators", or MACROPS [Fikes, Hart, and Nilsson 72]. A MACROP provides a short-cut through the search space -- if the system is so fortunate as to have previously solved a problem which is a current subtask. REFLECT systematically combined primitive operators into BIGOPS during a preprocessing stage. For example, an operator to pick up a block, and an operator to stack a block being held on top of another, are paired in the BIGOP "pickup^stack". If the planner only uses paired

BIGOPS, the search space is cut in half. MACROPS and BIGOPS are simply expanded at plan execution time.

MACROPS and BIGOPS preserve all the preconditions and effects of their component operators. [Sacerdoti 74] argues that the retention of detail greatly limits the heuristic value of such simplifying representations. A hierarchy of abstraction spaces should model the world and actions in less and less detail. A short, abstract plan can be quickly found at the least detailed level. Subtasks are created at the next level of detail to link the steps of the abstract plan. Only at the bottom of the hierarchy are all the facts known to the planner considered; but the search remains efficient, as it is tightly constrained by the more abstract plan.

ABSTRIPS [Sacerdoti 74] and NOAH [Sacerdoti 77] implemented a small portion of this philosophy by selectively ignoring action preconditions. In ABSTRIPS, preconditions were assigned criticality values. At abstraction level N, the operators are simplified by disregarding preconditions valued less than N. The planner repeatedly patches the abstract plan, so that it remains correct for the more and more detailed versions of the operators.

NOAH represents actions by procedures, and permits a dynamic hierarchy, but the approach is basically the the same: some preconditions are initially ignored, and the initial plan is heuristically patched up.

A "detail", however, is not necessarily an unimportant detail. The patching process can simply fail at any level. The planner

could try to come up with another abstract plan -- but would have no guarantee that this second attempt would not be unfixable as well.

Again, no formal description of the planning systems or the precise relationship between the levels of the hierarchy was ever presented.

2.4 PLANNING IN PROPOSITIONAL DYNAMIC LOGIC

Dynamic logic is a tool for reasoning about binary state-relationships induced by computer programs, developed by workers in program semantics and verification [Harel 79] [Pratt 76]. [Rosenschein 81] formulated planning in terms of propositional dynamic logic, explicitly equating the notions of "plan" and "program". Plans are allowed to contain tests and conditional sequences of actions.

A dynamic logic is a modal logic: a plan is a reachability relationship over a set of possible worlds. If A is a plan, [A] corresponds to a "necessary" modality; [A]p is true in a world I if p is true in every world reachable from I by A. So [A]p can be read, "after A, p". Similarly, <A> corresponds to a "possible" modality; <A>p is true in a world I if p is true in some world reachable from I by A. The fact that the box and diamond modalities are distinct intuitively implies that plans may be non-deterministic. When $p \supset [A]q$ holds, we say "p is a (sufficient, but perhaps not necessary) precondition of A and q" and "q is a postcondition of p and A".

A loop-free subset of propositional dynamic logic is defined as follows:

2.4.1 Syntax

A dialect of PDL is specified by its atomic propositions PROP and atomic actions ACT. Define Wffs and Plans over <PROP,ACT> inductively by:

1. If $p \in$ PROP then $p \in$ Wffs
2. If $a \in$ ACTS then $a \in$ Plans
3. If $p,q \in$ Wffs then $\neg p$, $p \vee q \in$ Wffs
4. If $p \in$ Wffs and $A \in$ Plans then $\langle A \rangle p \in$ Wffs
5. null $\in$ Plans
6. If $p \in$ Wffs and nonmodal, and $A,B \in$ Plans, then
   $A;B$, $A \cup B$, and $p? \in$ Plans

A wff is nonmodal if it contains no subformula of the form $\langle A \rangle p$. Abbreviate:

$\neg(\neg p \vee \neg q)$	as	$p \wedge q$
$\neg p \vee q$	as	$p \supset q$
$\neg\langle A \rangle\neg$	as	$[A]$
$(p?;A \cup \neg p?;B)$	as	$(p \rightarrow A,B)$

The form $(p \rightarrow A,B)$ represents the conditional, IF p THEN A ELSE B. The form $A;B$ represents the sequential execution of A then B. We write $A;B;C$ indifferently for $(A;B);C$ or $A;(B;C)$.

2.4.2 Semantics

The semantics described here are a subset of the first-order semantics presented in the next chapter, rather than those which appear in [Rosenschein 81]. A structure for PDL contains a set of worlds WD and a function m which interprets each atomic action as a binary relationship over WD (i.e., a set of pairs of worlds). Each

world assigns a truth value to the atomic propositions; if p holds in world I, write I|=p. Mutually define truth for non-atomic wffs in a world I, and a meaning function M for general plans:

Truth of formulas

$I|= \neg p$ iff not $I|=p$
$I|= p \vee q$ iff $I|=p$ or $I|=q$
$I|= \langle B\rangle p$ iff exists $J \in WD$ such that
$(I,J) \in M(B)$ and $J|=p$

Meanings of Programs

$M(a) = m(a)$ for $a \in ACT$
$M(null) = \{(I,I) \mid I \in WD\}$
(identity relation over WD)
$M(p?) = \{(I,I) \mid I|= p\}$
(identity relation restricted to worlds where p holds)
$M(A;B) = m(A) \circ m(B)$
(composition of relations)
$M(A \cup B) = m(A) \cup m(B)$
(union of relations considered as sets)

A structure satisfies a wff if the wff is true in every world in the structure. A valid wff is satisfied by all structures.

2.4.3 Axiomatics

Take A and B to be any Plans, p and q to be any Wffs. A complete axiomatization of uninterpreted loop-free PDL is given by:

Axioms

1. Axioms of propositional calculus
2. $[A](p \supset q) \supset ([A]p \supset [A]q)$
3. $[null]p \equiv p$
4. $[p?]q \equiv (p \supset q)$
5. $[A;B]p \equiv [A][B]p$
6. $[A \cup B]p \equiv ([A]p \wedge [B]p)$

Rules of Inference

From p, p⊃q derive q
From p derive [A]p

Write ⊢p if p is provable from these axioms, and G⊢p if p is provable from the axioms augmented by some set of wffs G.

2.4.4 Normal Form Plans

A plan in which the "?" (test) and "U" (non-deterministic choice) operators are only used to construct conditional forms (p → A,B) is a C-Plan. A plan B is in normal form if:

(i) B can be written as A1;A2;...;An with at most one Ai not atomic; and

(ii) Such an Ai is of the form (t → B1,B2), where t is an atomic proposition, and B1 and B2 are in normal form.

Any C-Plan is equivalent to a C-Plan in normal form. A set of meaning-preserving transformations that can be used to put any C-Plan in normal form is easily found; for example,

(¬p → A,B) becomes (p → B,A)

(p v q → A,B) becomes (p → A,(q → A,B))

(p → B1,B2);...;(q → B3,B4) becomes
p → (B1;...;(q → B3,B4)), (B2;...;(q →B3,B4))

We'll only look for normal form plans as solutions to planning problems. The following discussion is simplified by informally identifying the null action with a plan of length 0, and taking A;null = null;A = A for any plan A. (It is obvious, of course, that the interpretation of the three strings above is identical.)

2.4.5 Defining A Planning Problem

If r describes some initial state, and s describes some goal, the wff

$$r \supset [A]\ s$$

clearly asserts that A is a plan for achieving s given r. Similarly, if a is an atomic action, one may write

$$p \supset [a]\ q$$

to assert that p is a precondition for using a to achieve q. Rosenschein defines a planning problem as a triple <VOC,G,RES(u)> where

1. VOC = <PROP,ACT> is the vocabulary of the problem.

2. G is a finite set of axioms, the domain constraints. It contains:

   Gs, a set of nonmodal, or static constraints.

   Gd, a set of dynamic constraints, which specify (perhaps only in part) the effects of the atomic actions. Each dynamic axiom is of the form p ⊃[a] q, where p and q are nonmodal and a is atomic.

3. RES(u) is a finite set of formulae called the plan constraints. Each is of the form r ⊃[u] s where r and s are nonmodal. "u" is a special syntactic variable which does not appear in VOC.

A solution to such a planning problem is normal-form plan B (in the dialect VOC) such that substituting B for u in each of the plan constraints creates a formula derivable from the domain constraints. That is, G⊢ r ⊃[B] s for each formula r ⊃[u] s in RES. Solutions can also be characterized semantically: structures that satisfy the domain constraints also satisfy the B-instantiated plan constraints.

2.4.6 A Planning Algorithm

Rosenschein suggests that the search for solutions to a planning problem be organized by the syntactic structure of the planning (programming) language. Suppose we are looking for a normal form plan A that satisfies one of the plan constraints r ⊃[A]s in FES. Consider the following cases corresponding to the possible forms of A:

1. A = null. This is a solution if G⊢ r⊃s.

2. A = a;B or A = B;a for some atomic action a. In the former case, A is a solution if G⊢ r/a ⊃[B]s, where r/a represents the strongest provable postcondition of r and a. Analogously, in the second case, A is a solution if G⊢ r ⊃[B]a\s where a\s is the weakest provable precondition of a and s. We call the former case "progression" and the latter "regression".

3. A = t → B1,B2. In this case, A is a solution if G⊢ (r ∧ t)⊃[B1]s and G⊢ (r ∧ ¬t)⊃[B2]s.

Case (1) defines success, (2) suggests forward and backward strategies for sequential steps, and (3) suggests a forward strategy for conditionals.

There are a number of simple ways of pruning the search tree. First, if r⊃r/a, the forward search need not consider action a. (Intuitively, a would not achieve anything that does not already hold in r.) Dually, if a\s⊃s, the backward search need not consider a. (a\s describes a goal which is harder to achieve than s -- perhaps impossible, when a\s is false.) These check eliminate self-loops; as described in chapter 4, they can be extended to eliminate all loops in the search space. Finally, if r⊃t or r⊃¬t, it is not necessary to pursue the forward conditional search involving t: the test uncovers no new information.

These observations lead to the following nondeterministic, bidirectional search algorithm.* The single constraint r⊃[u]s is solved for by calling BIGRESS(r,s).

BIGRESSION ALGORITHM

```
BIGRESS(r, s):

   If G⊬ r⊃s then return( null ).

   Choose:

      Choose <a, r/a> from LiveForward(r);
         return( a;BIGRESS( r/a, s ) ).

      Choose <a, a\s> from LiveBackward(s);
         return( BIGRESS( r, a\s );a ).

      Choose t from NonTriv(r);
         return( t → BIGRESS( r ∧ t,  s ),
                     BIGRESS( r ∧ ¬t, s ) ).
end.
```

```
LiveForward(r):
   return { <a, r/a> | a ∈ ACT, G⊬ r⊃r/a}.


LiveBackward(s):
   return { <a, a\s> | a ∈ ACT, G⊬ a\s⊃s}.


NonTriv(r):
   return { t | t ∈ PROP, G⊬ p⊃t, G⊬ p⊃¬t}
```

The algorithm still presents a fairly abstract view of the planning process; it says nothing about the particular order in which "choices" are made, when backtracking is invoked, etc. (If one of the subprocedures returns an empty set at some point, that choice is obviously prohibited.) The extensions to BIGRESS to handle multiple

---

*Rosenschein's original statement of the algorithm differed by using two additional parameters to pass the partially constructed plan to each recursive instance of BIGRESS. The innermost instance returned the entire plan.

constraints are discussed along with the first-order extensions in chapter 4. We now discuss how the provability test, "G⊢", and the progression and regression functions, "/" and "\", can be implemented.

2.4.7 Provability, Progression, And Regression

It is notoriously difficult to mechanize modal proof systems; the auxiliary functions are implemented using only ordinary propositional decision methods. The key requirement is that the nonmodal axioms, Gs, be strong enough to generate all nonmodal formulae generated by all of G. Rosenschein does not present a proof that this condition can always be enforced, while Gs remains finite. Section 4.3 discusses some of difficulties that arise in the first-order case. We simply take it on faith that this requirement is met.

The test "G⊢" is only applied to nonmodal formulas p. Given the above caveat, this is equivalent to Gs⊢p, which can be checked by a standard propositional theorem prover.

Now, quoting [Rosenschein 81]:

> The formula r/a is found by taking the conjunction of the set of formulae each of which is a disjunction of a set of qi taken from the "right hand side" of the dynamic axioms of G (pi⊃[a]qi) such that the disjunction of the corresponding pi's is implied by r. Dually, a\s is found by taking the disjunction of the set of formulae each of which is a conjunction of a set of pi drawn from the "left hand side" of the dynamic axioms of G such that the conjunction of the corresponding qi's implies s.

Consider the following sample axioms:

$$\begin{array}{lll} A & \supset[a] & (B \vee C) \\ G & \supset[a] & \neg B \\ (F \wedge E) & \supset[a] & D \end{array}$$

In this case, $a\backslash(C \vee D) = (A \wedge G) \vee (F \wedge E)$. The reason for this is that $(B \vee C)$ conjoined with $\neg B$ implies $(C \vee D)$, so the conjunction of the corresponding left-hand sides $(A \wedge G)$ is one disjunct of $a\backslash(C \vee D)$. Likewise, the formula D alone implies $(C \vee D)$, making the corresponding left-hand side $(F \wedge E)$ the second disjunct. These two cases exaust the possibilities for getting $(C \vee D)$.

Rosenschein does not present a proof that "/" and "\" as outlined do indeed calculate the strongest provable postcondition and weakest provable precondition. Section 4.5 of this thesis shows the value of r/a is a post condition, but still lacks a proof that it is the strongest postcondition; and similarly for a\s. Nonetheless, "/" will continue to be used for stongest provable postcondition, and "\" for weakest provable precondition.

The description of "/" and "\" suffices in the propositional framework because there is only a finite number of dynamic axioms. One may simply form all possible disjunctions of the "left hand sides", for example, and test whether they are implied by s. In the first-order case, we have to deal with dynamic axioms containing quantifiers, and explicitly discuss how formulae containing variables are progressed and regressed. In order to extend the propositional techniques, we must be able to determine all the relevant "instances" of the quantified dynamic axioms; the main work of this thesis is to show how a proof method known as "answer extraction" can be employed to this end.

2.4.8 Hierarchical Planning

The key observation in extending the single-level algorithm to multi-level, hierarchical planning is that an atomic action at level k is a plan to be solved at level k+1. This point of view is possible because of the way the planning problem was formalized. The syntactic form of a dynamic axiom in Gd is precisely the same as a plan constraint in RES.

Formally, a hierarchical planning problem is a tree of single-level problems. If $<VOC_k = <PROP_k, ACT_k>, G_k, RES_k(u_k)>$ is the problem at non-leaf node k, then node k has one successor for each $a_{k,i}$ in ACT_k, and that successor's problem has the form $<VOC_{k+1}, G_{k+1}, G'(a_{k,i})>$ where G' denotes the subset of dynamic axioms of G_{k+1} having the form $p \supset [a_{k,i}]q$. In other words, the domain constraints on the primitive "a" at level k become plan constraints at level k+1. An overall solution is then a plan using the vocabulary of the leaf nodes that satisfies the requirements of the root node.

For any node k, only the successor nodes corresponding to actions actually used in the solution need be solved. Furthermore, the existence of a solution for each of these nodes guarantees the existence of an overall solution. The relationship between levels in the hierarchy is <u>precisely</u> defined as "correctly implements".

2.4.9 Summary

Like the situational calculus, Rosenschein's framework provides both a proof theory and semantics for planning (and, as well, is forced to deal with explicit frame axioms). As in the state-space planners, search is separated from theorem proving, which is hidden in auxiliary functions. The BIGRESS algorithm provides a "hook" for adding search heuristics. The limitations of STRIPS type operators are abandoned. The general progression and regression operators incorporate the model-updating operations of STRIPS and its progeny as special cases. Bidirectional search can offer considerably better performance than unidirectional search [Nilsson 80]. Sacerdoti's "hierarchy of abstraction spaces" is given a formal definition. Not only may actions be described in greater or lesser detail at the various levels, but entirely different actions may appear, and the world may be described using different vocabularies. The exact hierarchy prevents unhappy clashes between subplans, as does the exact hierarchy obtained by the use of MACROPS and BIGOPS.

It must be admitted that some of the comparisons with earlier systems are not really fair: BIGRESS gains much of its generality (as in it's handling all kinds of disjunctive and conjunctive goals and state descriptions) by avoiding some of the particular issues of efficiency which led to STRIPS. In any case, the descriptive power of dynamic logic makes it a powerful tool for formally comparing and evaluating various planning strategies.

CHAPTER 3

A FIRST-ORDER DYNAMIC LOGIC

3.1 THE LANGUAGE

There is no one first-order dynamic logic; many logics have been defined which fall under this rubric [Harel 79]. The basic plans , or actions, with which workers in program verification are concerned relate worlds with differing interpretations of some functions (i.e. program variables); we are instead most interested in actions which affect only the extensions of predicates (e.g., the truth value of on(BLOCK9, TABLE) is changed by the action pickup(BLOCK9)). The rest of this chapter gives the syntax, semantics, and axiomatics of a language I call Transparent Dynamic Logic, designed to express just such actions. The general outline follows [Harel 79], but the particular formulation of actions with parameters and transparency conditions is original.

3.2 SYNTAX

Define the sets of symbols:

VAR=variables

CON=constants

PRED(j)=predicate symbols of arity j, for j=0,1,2,...
PRED(2) includes at least "=".

ACT(j)=action symbols of arity j, for j=0,1,2,...

Let TERMS=VAR u CON
PRED=PRED(0) u PRED(1) u ...
ACT=ACT(0) u ACT(1) u ...

The well-formed formulas and plans of TDL are defined by mutual recursion.* An atomic formula is an n-ary predicate symbol followed by n terms. An atomic action is an n-ary action symbol followed by n terms. The terms are the parameters of the action.

1. An atomic action is a TDL-plan.

2. Where P is a non-modal TDL-wff, and A,B are TDL-plans, then null, (A;B), (A U B), and P? are TDL-plans.

3. An atomic formula is a TDL-wff.

4. Where x ϵ VAR, P,Q ϵ TDL-wffs, and A ϵ TDL-plans, then $\neg$P, (PvQ), E x P, and <A>P are TDL-wffs.

A non-modal TDL-wff is a TDL-wff containing no TDL-plan. (I.e., a formula of function-free first-order logic.) We make the following abbreviations:

---

* Note on typography: "v" is or; "$\wedge$" is and; "$\neg$" is not; "$\supset$" is implies; "$\equiv$" is two-way implication; "ϵ" is member; "u" is set union; "$\subseteq$" is subset; "E" is the existential quantifier; "$\forall$" is the universal quantifier.

[A]	for ¬<A>¬
∀x	for ¬E x¬
P ∧ Q	for ¬P ∨ ¬Q
P ⊃ Q	for ¬P ∨ Q
P ≡ Q	for (P ⊃ Q)∧(Q ⊃P)
t1≠t2	for ¬(t1=t2)
true	for p ∨ ¬p
false	for p ∧ ¬p
(P →A,B)	for ((P?;A) U (¬P?;B))

The plan (P → A,B) corresponds to the more familiar programming language construct IF P THEN A ELSE B.

A list of variables x1,...,xn will often be abbreviated as X, as in

(E X) P for Ex1 Ex2 ... Exn P

In general, a capital letter standing in a place where a variable is expected will stand for a list of variables. Constants will always be capitalized.

A particular dialect of TDL is specified by its vocabulary, <CON, PRED, ACT>.

A plan B is in normal form if:

(i) B can be written as A1;A2;...;An with at most one Ai not atomic; and

(ii) Such an Ai is of the form (TEST → B1,B2), where TEST is a nonmodal wff and B1 and B2 are in normal form; and

(iii) Put in prenex form, TEST's quantifiers are either all ∀ or all E.

Unlike the propositional case, TEST cannot always be made atomic. The need for condition (iii) is discussed in section 4.4.

3.3 SEMANTICS

A structure, or interpretation, of TDL is a triple <DOM, WD, m> where DOM is a domain, WD a set of worlds, and m is a function which interprets the action symbols. Each world interprets the terms as members of the domain, and the predicate symbols as predicates over the domain. In an acceptable transparent structure:

1. The meaning of "=" is fixed as equality.

2. Constants receive the same interpretations in all worlds.

3. For any world I, variable x, and entity d, W contains $\{d/x\}I$, a world just like I except that x is interpreted as d.

4. Actions are transparent, as explained below.

A meaning function M interprets a plan as a binary relationship over WD (i.e., a set of pairs of elements of WD), using m to map action symbols to functions from entities to binary relations over WD. We define M and truth in a world I, written $I\models$, as follows.

$I\models p(t1,\ldots,tn)$ iff $I(p)(I(t1),\ldots,I(tn))$
where $I(p)$ is the interpretation of predicate symbol p and $I(ti)$ is the interpretation of term ti in world I.

$I\models t1=t2$ iff $I(t)=I(t2)$

$I\models \neg P$ iff not $I\models P$

$I\models P \vee Q$ iff $I\models P$ or $I\models Q$

$I\models \exists x\, P$ iff for some $d \in DOM$, $\{d/x\}I\models P$

$I\models \langle A\rangle P$ iff for some $J \in WD$, $(I,J) \in M(A)$ and $J\models P$

Define $M: \text{TDL-plans} \rightarrow 2^{WxW}$, where

$m: ACT(j) \rightarrow (: DOM^{j} \rightarrow 2^{WxW})$

as follows:

$$M(a(t1,\ldots,tn)) = \{(I,J) \mid (I,J) \in m(a)(I(t1),\ldots,I(tn))\}$$

$$M(null) = \{(I,I) \mid I \in WD\}$$

$$M(A \cup B) = M(A) \cup M(B) = \{(I,J) \mid (I,J) \in M(A) \text{ or } (I,J) \in M(B)\}$$

$$M(P?) = \{(I,I) \mid I \models P\}$$

$$M(A;B) = M(A) \circ M(B) = \{(I,J) \mid \text{Exist } K \in WD \text{ such that } (I,K) \in M(A) \text{ and } (K,J) \in M(B)\}$$

Condition (2) requires that

$$I(C)=J(C), \text{ all } I,J \in WD, C \in CON$$

The null action simply takes any world to itself, while a test action P? is the identity relation restricted to worlds where P is true. One can derive semantics for the normal conditional form:

$$M(P \rightarrow A,B) = \{(I,J) \mid \text{Either } I \models P \text{ and } (I,J) \in M(A) \text{ or } I \models \neg P \text{ and } (I,J) \in M(B)\}$$

The definition of M for atomic actions indicates that parameters are evalutated in the initial world. Condition (3) above insures that all quantified formulas can be interpreted.

The transparency conditions on actions allow substitution of terms through modal contexts. For example, from

$$\forall x(box(x) \supset [pickup(x)]hold(x))$$

we'd like to conclude

$$box(B1) \supset [pickup(B1)]hold(B1)$$

but nothing so far assures that x has the same interpretation before and after pickup(x), and that the only input to pickup is its

parameter. The requirement is:

(I,J) ϵ M(A) $\supset$ I(t)=J(t), any term t, any atomic action A

(I,J) ϵ M(a(t1,...tn)) $\supset$
([d/x] I, [d/x] J) ϵ M(a(t1,...,tk))
for any d ϵ DCM, x ϵ VAR,
whenever x is not one of t1,...,tk

The result of all this is that only quantifiers will bind variables.

Say S is a structure and P is a TDL-wff. S satisfies P if P is true in every world in S. P is valid, written

|= P

if every transparent structure satisfies P. Where G is a set of formulas (axioms), we write

G |= P

to mean that every transparent structure which satisfies all of G satisfies P.

3.4 AXIOMATICS

The axioms and rules of inference include all those for function-free, first-order logic with equality and loop-free dynamic logic. P and Q will stand for any TDL-wff, and A for any atomic action. "The term t is a parameter of atomic action A" is abbreviated t ϵ A. The usual notions of bound and free occurences of a variable hold. For example, x is free in the wff

[drop(x)] onfloor(x)

We write

P[t/x]

for the formula obtained by substituting term t for all free

occurences of x in formula P:

$x\{t/x\} = x$
$s\{t/x\} = s$ for any term s other than x
$p(t1,...,tn)\{t/x\} = p(t1\{t/x\},...,tn\{t/x\})$
$(\neg P)\{t/x\} = \neg(P\{t/x\})$
$(P \vee Q)\{t/x\} = (P\{t/x\} \vee Q\{t/x\})$
$(\forall y P)\{t/x\} = \forall z((P\{y/z\})\{t/x\})$ new variable z
$([a(t1,...,tn)]P)\{t/x\} =$
$[a(t1\{t/x\},...,tn\{t/x\})]\ P\{t/x\}$

first order

A1. All tautologies of propositional calculus

A2. $\forall x(P \supset Q) \supset (\forall x P \supset \forall x\ Q)$

A3. $\forall x P \supset P\{t/x\}$, for any term t

A4. $P \supset \forall x P$, where x is not free in P

modal

A5. $[A](P \supset Q) \supset ([A]P \supset [A]Q)$

A6. $[P?]Q \equiv (P \supset Q)$

A7. $[null]P \equiv P$

A8. $[A][B]P \equiv [A;B]P$

A9. $[A \cup B]P \equiv [A]P \wedge [B]P$

A10. $\forall x[A]P \supset [A]\forall x P$, if $x \notin A$

equality

A11. $\forall x(x=x)$

A12. $t1=t2 \supset (p(...,t1,...) \supset p(...,t2,...))$

all $p \in PRED$

A13. $\forall xy(x=y \supset [A]\ x=y)$

A14. $\forall xy(x \neq y \supset [A]\ x \neq y)$

rules of inference

R1. From P, P⊃Q conclude Q

R2. From P conclude ∀xP

R3. From P conclude [A]P

The axiomatization is sound, where we take soundness to mean

If G⊢P then G|=P

The only non-standard axioms are A10 and our version of A3 with substitutions allowed through atomic actions; the transparency conditions enforce the truth of these axioms. A10, the so-called Barcan formula of modal logic, also depends upon our definition of a structure as containing a single domain. A10 would not hold if we ever wished to model actions which actually create and destroy entities.

The axiomatization is complete in the sense that

If G|=P then G⊢P

Since we only give partial semantics for the atomic actions, we won't talk about completeness in terms of capturing all the true formulas in a particular structure. ([Harel 79] and [Pratt 76] demonstrate complete axiomatizations of loop-free first-order dynamic logic where the only primitive actions are test and assignment.)

3.5 LEMMAS

Some useful derived rules of inference are:

DR1: From P⊃Q conclude ∀xP ⊃ ∀xQ

DR2: " ExP ⊃ ExQ

DR3: " $[A]P \supset [A]Q$

DR4: " $\langle A\rangle P \supset \langle A\rangle Q$

The following simple theorems of TDL shall prove helpful as well.

TH1: If P is a TDL-wff and a valid formula of first-order logic with equality, then $\vdash P$.
Proof: Obvious, since TDL incorporates FOLE.

TH2: If $G\vdash P\supset Q$ then $Gu\{P\}\vdash Q$.
Proof: R1 applied to $Gu\{P\}\vdash P$, $Gu\{P\}\vdash P\supset Q$.
The reverse does not hold.

Lemma: $\vdash Ex[A]P \supset [A]ExP$, if $x \notin A$.
Proof:

1.	$\forall x\neg P \supset \neg P$	A3
2.	$\forall x\langle A\rangle\forall x\neg P \supset \forall x\langle A\rangle\neg P$	1, DR4, DR1
3.	$\langle A\rangle\forall x\neg P \supset \forall x\langle A\rangle\forall x\neg P$	A4
4.	$\langle A\rangle\forall x\neg P \supset \forall x\langle A\rangle\neg P$	2, 3, PC
5.	$Ex[A]P \supset [A]ExP$	4, PC

TH3: $\vdash K[A]P \supset [A]KP$
where K is a sequence of quantifiers, and none of the quantified variables appear in A.
Proof: Induction on the length of K, using A10 and TH2.
This theorem allows us to "push" quantifiers through actions.

TH5: $([A]P \wedge [A]Q) \supset [A](P \wedge Q)$
TH6: $([A]P \vee [A]Q) \supset [A](P \vee Q)$
Proof: These are true for any dynamic logic; see [Harel 79].

CHAPTER 4

A FIRST-ORDER FRAMEWORK

We now generalize the propositional framework described in the chapter 2 to handle a subset of first-order planning problems. BIGRESS is extended to plan to satisfy multiple plan constraints. The notion of planning to satisfy multiple constraints is more general than that of planning to achieve multiple (conjunctive) goals, and will prove useful in the later discussion of hierarchical planning. The progression and regression functions are discussed in detail. Finally, some comments are made on suitable control strategies for the non-deterministic BIGRESS algorithm.

4.1 FORMULATION OF A PLANNING PROBLEM

Since TDL incorporates first-order logic, it obviously not decidable. It is well-known, however, that the validity problem is decidable for first-order formulas which can be put in prenex form so that all universally-quantified variables preceed all existentially-quantified ones [Ackermann 1954]. We restrict our attention to a class of planning problems which can be solved using

only first-order reasoning about formulas of this form.

A prenex wff or set of wffs which in all ∀-quantifiers appear before all E-quantifiers is "∀-first"; the converse is "E-first".

A planning problem is a triple <VOC,G,RES(u(H))> where:

1. VOC = <CON,PRED,ACT> is the vocabulary of the problem: the particular dialect of TDL to be used in synthesizing a solution. The vocabulary (in particular, CON) is finite.

2. G is a set of axioms, the domain constraints. It contains:

Gs, the static axioms, a finite set of E-first non-modal formulas.

Gd, the dynamic axioms, a finite set of formulas of the form:

$$(\forall Y)(p \supset [a(Z)]\ q)$$

Y is a list of variables and Z is a list of terms such that Y contains any variables in Z; p and q are non-modal quantifier-free formulas all of whose free variables are contained in Y. Where Z is of length j, a ∈ ACT(j).

Ge, a special set of axioms of the form

$$C1 \neq C2$$

for all C1, C2 ∈ CON, C1 not the same as C2.

3. RES(u(H)) is a set of k plan constraints, each of the form

$$(\forall X)(r \supset [u(H)]\ s)$$

X is a list of variables and H a list of terms such that X contains any variables in H; r is a non-modal, E-first wff; s is a non-modal, ∀-first wff; and the free variables in r and s fall in

X. The variables in H are called the problem parameters.

A solution is a normal-form plan B such that substituting B for u(H) in each plan constraint creates a theorem derivable from G, and the only free variables in B are the problem parameters. H may be thought of as the "input" to the plan.

For example, if RES(u(h)) is

$(\forall\ h)(block(h) \supset[u(h)]\ inbox(h))$

$(\forall\ h\ x)((\neg inbox(x) \wedge x \neq h) \supset[u(h)]\ \neg inbox(x))$

the constraint is to find a plan to put any block h into the box, without putting anything else into the box. The only input to the plan will be the particular block to be moved.

Constants are made pairwise disjoint mainly as a matter of convenience; in the examples which will be presented, free variables are used in place of non-unique constants. We are not assuming that there is a constant for every entity in the domain (unless, of course, Gs specifically includes a domain closure axiom).

We define a frame axiom as a dynamic axiom

$(\forall\ Y)(p \supset[a(Z)]\ q)$

such that

$G \vdash (\forall\ Y)(p \supset q)$

A pure frame axiom is one of the form

$(\forall\ Y)(p \supset[a(Z)]\ p)$

A predicate p ∈ PRED(j) is <u>invariant</u> in G if for any a ∈ ACT(k), lists of variables Y and Y',

G⊢ (∀ Y Y')(p(Y) ⊃ [a(Y')] p(Y))

G⊢ (∀ Y Y')(¬p(Y) ⊃ [a(Y')] ¬p(Y))

hold. Note that "=" is invariant.

4.2 BIGRESS ALGORITHM

We review and extend Rosenschein's bigression algorithm.

The function "/" calculates the strongest provable postcondition of a condition and an action, "\" calculates the weakest provable precondition, and "G⊢" takes as input a non-modal formula P, and decides whether P is provable from G. Where RES(u(H)) is the set of wffs

{ (∀ X1)(r1 ⊃ [u(H)] s1), ... , (∀ Xk)(rk ⊃ [u(H)] sk) }

A is an atomic action, and TEST a wff, define

R = {r1,...,rk}
S = {s1,...,sk}
R/A = {r1/A,...,rk/A}
A\S = {A\s1,...,A\s1}
R ∧ TEST = {r1 ∧ TEST,...,sk ∧ TEST}

An acceptable plan is returned by calling BIGRESS(R,S). Assume dynamic scoping of R and S for recursive calls within BIGRESS.

```
BIGRESS(R,S):

   If G⊢ ri ⊃ si for 1<=i<=k
        then return( null ).

   Choose:

      Choose <A, R/A> from Liveforward(R):
         return( A;BIGRESS( R/A, S ) ).

      Choose <A, A\S> from Livebackward(S):
         return( BIGRESS( R, A\S );A ).

      Choose TEST from NonTriv(R):
         return( TEST → SubPlan1, SubPlan2 ) where
            SubPlan1:= BIGRESS( R ∧ TEST, S);
            SubPlan2:= BIGRESS( R ∧ ¬TEST, S).
end.



Liveforward(R):
   return({ <a(t1,...,tj), R/a(t1,...,tj)> |
         a ∈ ACT(j), ti ∈ CON ∪ H for 1<=i<=j, and
         for SOME ri ∈ R,
         not G⊢ ri ⊃ ri/a(t1,...,tj) }).



Livebackward(S):
   return({ <a(t1,...,tj), a(t1,...,tj)\S> |
         a ∈ ACT(j), ti ∈ CON ∪ H for 1<=i<=j, and
         for SOME si ∈ S,
         not G⊢ a(t1,...,tj)\si ⊃ si }).



NonTriv(R):
   return({ TEST | TEST is a non-modal TDL-wff,
         all free variables of TEST are in H,
         TEST's quantifiers are either all ∀ or all E,
         and for SOME ri ∈ R,
         not G⊢ ri ⊃ TEST
         not G⊢ ri ⊃ ¬TEST }).
```

Attempting to choose from an empty set is equivalent to failing. It may be desirable to impose further restrictions on

tests: for example, only allow predicates which directly correspond to the robot's sensory inputs. Thus, "handempty" might be a legitimate test, but not "inside(BLK1,BOX2)" since BLK1 would be hidden from view.

4.3 CORRECTNESS AND COMPLETENESS OF BIGRESS

Define correctness and completeness of the implementations of the auxiliary functions "G⊢", "/", and "\" as follows:

For any atomic action A, non-modal wffs p, r, s:

1. Correctness:

   a) If "G⊢p" returns true then p follows from G.

   b) G⊢ r ⊃ [A] r/A

   c) G⊢ A\s ⊃ [A] s

2. Completeness:

   a) If p follows from G then "G⊢p" returns true.

   b) If G⊢ r ⊃ [A] s then G⊢ r/A ⊃ s

   c) If G⊢ r ⊃ [A] s then G⊢ r ⊃ A\s

Definition: BIGRESS is correct if it only returns solutions; i.e., for any plan E returned by BIGRESS(R,S),

G⊢ (∀ Xi)(ri ⊃ [E] si)

for each wff (∀ Xi)(ri ⊃ [u(H)] si) ∈ RES(u(H)), and the only free variables in B are in H.

The definition of the completeness of BIGRESS uses the notions of an essential solution to a planning problem, and of two solutions being essentially the same.

Definition: B is an essential solution to a planning problem if any normal-form plan B' formed from B by deleting any atomic actions, and/or replacing any conditional form (TEST → B1,B2) by one of its branches B1 or B2 is not a solution.

Definition: A plan B is essentially the same as a plan B' if B' can be obtained from B by inserting and/or deleting null actions.

Definition: BIGRESS is complete if it can find a solution which is essentially the same as any normal-form solution B to a given planning problem.

We prove the correctness and completeness of BIGRESS relative to the correctness and completeness of "G⊢", "/", and "\". Sections 4.4 and 4.5 demonstrate the correctness of the suggested implementations of the auxiliary functions, and discuss some problems that arise in trying to obtain completeness. No actual proof of completeness is given.

4.3.1 Correctness Of BIGRESS

Theorem: Given the correctness of "G⊢", "/", and "\" (1(a)-(e) above), BIGRESS is correct. Proof:

Let B be a plan returned by BIGRESS(R,S). Its length |B| is the number of non-null tests and actions in B. Attach the phrase "for all i, 1<=i<=k" after each formula involving ri or si in the following.

If |B|=0, then B=null. The stopping test must have succeeded on the first call to BIGRESS, so

$G \vdash r_i \supset s_i$

By A7,

$G \vdash r_i \supset [null]s_i$

Otherwise suppose that BIGRESS is correct for plans shorter than |B|. On the first call to BIGRESS, the algorithm must have chosen one of:

(i) <A, R/A> from Liveforward(R).
BIGRESS(R/A,S) returns a plan B' such that

1.	$r_i/A \supset [B']s_i$	induction hypo.
2.	$[A]r_i/A \supset [A][B']s_i$	1,DR3
3.	$r_i \supset [A]r_i/A$	"/" correctness
4.	$r_i \supset [A;B']s_i$	2,3,A8

and B=A;B'.

(ii) <A, A\S> from Livebackward(S).
BIGRESS(A\S,S) returns B' such that

1.	$r_i \supset [B']A\backslash s_i$	ind. hypo.
2.	$A\backslash s_i \supset [A]s_i$	"\" correctness
3.	$[B']A\backslash s_i \supset [B'][A]s_i$	2,DR3
4.	$r_i \supset [B';A]s_i$	1,3,A8

and B=B';A.

(iii) TEST from NonTriv(R).
Then

1.	$(r_i \wedge TEST) \supset [SubPlan1]s_i$	ind. hypo.
2.	$(r_i \wedge \neg TEST) \supset [SubPlan2]s_i$	ind. hypo.
3.	$r_i \supset ((TEST \supset [SubPlan1]s_i) \wedge (\neg TEST \supset [SubPlan2]s_i))$	1,2,PC
4.	$r_i \supset ([TEST?;SubPlan1]s_i \wedge [\neg TEST?;SubPlan2]s_i)$	3,A6
5.	$r_i \supset [TEST?;SubPlan1 \cup \neg TEST?;SubPlan2]s_i$	4,A9

That is, $r_i \supset [B]s_i$ where $B = TEST \rightarrow SubPlan1, SubPlan2$

So in any case, B is a plan such that

$G \vdash r_i \supset [B]s_i$

By R3,

$G \vdash (\forall x_i)(r_i \supset [B]s_i)$

Therefore B is a solution to the planning problem

<VOC, G, { (∀ X1)(r1 ⊃ [u(H)] s1), ... ,

(∀ Xk)(rk ⊃ [u(H)] sk) } >

so BIGRESS is correct.

4.3.2 Completeness Of BIGRESS

Theorem: Given correctness and completeness of "G⊢", "/", and "\", BIGRESS is complete. Proof:

Let B be an essential solution to the planning problem

<VOC, G, { (∀ X1)(r1 ⊃ [u(H)] s1), ... ,

(∀ Xk)(rk ⊃ [u(H)] sk) } >

We show that BIGRESS(R,S) can return a plan essentially the same as B. Again, append "for all i, 1<=i<=k" after each formula below. First, note that if

G⊢ (∀ Xi)(ri ⊃ [B]si)

then by A3

G⊢ ri ⊃ [B] si

If |B|=0, then B=null; so G⊢ri⊃[null]si. By A7,

G⊢ ri ⊃ si

which is precisely the stopping test in BIGRESS: so null is returned.

So assume completeness holds for plans shorter than B. Where B',B1,B2 are plans, A an atomic action, and TEST a non-modal wff all of whose quantifiers are E or ∀, B is of one of the forms:

(i) B= A;B'
By A8
G⊢ ri ⊃ [A][B']si
and by "/" completeness

$G \vdash ri/A \supset [B']si$

Suppose <A, R/A> was not in Liveforward(R). Then

$G \vdash ri \supset ri/A$

which implies

$G \vdash ri \supset [B']si$

This would mean that B' is a solution to the planning problem, so B would not be an essential solution. Therefore, BIGRESS can choose <A, R/A>. By the induction hypothesis, the recursive call BIGRESS(R/A,S) can return a plan B'' which is essentially the same as B'. Then the first call to BIGRESS returns A;B'', which is essentially the same as B.

(ii) B= B';A

From

$G \vdash ri \supset [B'][A]si$

proceed to deduce

$G \vdash ri/B' \supset [A]si$	"/" completeness
$G \vdash ri/B' \supset A\backslash si$	"\" completeness
$G \vdash [B']ri/B' \supset [B']A\backslash si$	DR3
$G \vdash ri \supset [B']ri/B'$	"/" correctness

Therefore

$G \vdash ri \supset [B']A\backslash si$

As before, it must not the case that

$G \vdash A\backslash si \supset si$

for then B' would be a solution. BIGRESS can choose <A, A\S> from Livebackward; by the induction hypothesis, the call BIGRESS(R,A\S) can return B'' essentially the same as B'; and BIGRESS(R,S) returns B'';A essentially the same as B.

(iii) B= (TEST→ B1,B2)

Given that

$G \vdash ri \supset [TEST \rightarrow E1, B2]si$

A6, A9, and propositional manipulations let us conclude

$G \vdash (ri \wedge TEST) \supset [B1]si$

$G \vdash (ri \wedge \neg TEST) \supset [B2]si$

Suppose TEST were not in Nontriv(R). Then

$G \vdash ri \supset TEST$ or $G \vdash ri \supset \neg TEST$

But in the first case, $(ri \wedge TEST) \equiv ri$, so

$G \vdash ri \supset [B1]si$

--i.e., B1 would be a solution. Similarly, in the second case, B2 would be a solution. So TEST must be in Nontriv(R).

Finally, BIGRESS(R∧TEST,S) can return B1' essentially the same as B1, BIGRESS(R∧¬TEST,S) can return B2' essentially the same as B2, so BIGRESS(R,S) can return (TEST→ B1',B2') essentially the same as B.

Note that for some particular rh in R, it may be the case that $G \vdash rh \supset \neg TEST$, so rh ∧ TEST is false. So some of the wffs in R on a recursive call to BIGRESS may be false. Any plan will then satisfy that component of the constraint.

(iv) B= A
The plan A is essentially the same as A;null. By case (i) BIGRESS can find a plan essentially the same as A;null and thus essentially the same as A.

In any case, BIGRESS can return a plan essentially the same as B. By the above definition, BIGRESS is complete, relative to the completeness of "⊢", "/", and "\".

4.3.3 Partial And Tctal Correctness

Discussions of the correctness of programs distinguish the notions of partial and total correctness. The plan constraints and dynamic axioms we have used have all been in the form of partial correctness assertions:

r ⊃ [B] s

s will hold in any state reached by executing B in a state where r holds. In particular, the assertion is true if no state is reachable by B from one where r holds.

What is desired, however, are plans which are <u>totally</u> correct: executing B in a state where r holds takes one to a state where s holds; that is, partial correctness plus guaranteed termination of B. [Rosenschein 81] remarks that termination of a plan (program) B is asserted by

|= <B> true

From any state, executing B <u>can</u> take you to some state.

Although this definition happens to suffice for normal-form plans, it is not always adequate. [Harel 79] points out that the meaning of termination depends upon the intended method of plan

execution. For a simple sequential regime, it is required that all execution paths through B terminate. For instance, when

B= (null;false?) U (null;true?)

then

<B> true

but the execution path through the first half of the union reaches a dead-end at the false? test.

The restriction of ? and U to the if-then-else form ensures that whenever a test fails, a non-failure alternative (i.e., the one headed by the negation of the test) is immediately available. Thus, dead-ends can arise in a normal form plan only if it contains some atomic action which fails to terminate.

So, we require that any atomic action, instantiated with whatever parameters, always leads to at least one state. Semantically,

$$\forall\, a \in ACT(j)$$
$$\forall\, d1,\ldots,dj \in DOM$$
$$\forall\, I \in WD$$
$$\exists\, J \in WD \text{ such that } (I,J) \in m(a)(d1,\ldots,dj)$$

The axioms are augmented by:

A15. <A> true , any atomic action A

A simple inductive argument then shows that $\vdash$<B>true for any normal form plan B.

4.4 THE PROVABILITY TEST

"G⊢" can be implemented by a complete first-order theorem prover if the non-modal axioms, Gs and Ge, are strong enough to derive all non-modal formulas derivable from G. When this is the case, tests of the form

G⊢ P , P non-modal

are equivalent to tests of the form

⊢ (Gs ∧ Ge) ⊃ P

When P is ∀-first and Gs E-first, the expression above is ∀-first. The decidabilty of this form is apparent in light of Skolem's theorem. The formula is valid iff M, the skolem form of its negation, is inconsistent. M can contain only nullary function symbols; thus, the Herbrand universe for M will be finite. All ground instances of M over this universe can be generated and tested for inconsistency.

Care was taken in BIGRESS to ensure that the provability test was only applied to ∀-first formulas. Tests were restricted to E-only or ∀-only forms so that both TEST and ¬TEST would be ∀-first.

Rosenschein dismisses as "pathological" cases in the propositional framework where non-modal theorems are derivable from G but not Gs. His example is the dynamic axiom

p ⊃ [a]false

which together with <a>true (A15) allows one to conclude ¬p.

The problem seems more tangible in the first-order framework. Say the dynamic axioms include:

∀ y (p(y) ⊃ [a] q(y))

$$\forall y\ (r(y) \supset [a] \neg q(y))$$

With A15 we can derive $\forall y(p(y) \supset \neg r(y))$.

In the propositional case and many instances of the first-order case, it is possible to mechanize the process for finding axioms which need to be added to Gs. Look for inconsistent sets of dynamic axiom consequences, and add as a new axiom the negation of the conjunction of the corresponding antecedents.

It is not always possible, however, in the first-order framework, to make Gs as strong as G with respect to non-modal theorems. The first example in section 5.2 gives a case where the dynamic axioms force the domain to be infinite but such a condition cannot be specified by a finite number of E-first non-modal axioms.

In brief: the implementation of "G⊢" that uses only Gs and Ge is correct (as defined in 4.3.2), but in some cases not complete. The progression and regression algorithms also assume that the non-modal axioms suffice to prove non-modal formulas, and similarly are correct but not necessarily complete.

4.5 PROGRESSION AND REGRESSION

The progression and regression functions generalize the propositional functions outlined in section 2.4.7. Recall that the progression of a state description through an action was computed by finding all the combinations of antecedents of the dynamic axioms which were implied by the description, and taking the conjunction of the corresponding dynamic axiom consequences. In the first-order

case, a single dynamic axiom may have a number of instances which are relevant to computing the progression: for example, $\forall y(p(y) \supset [a]q(y))$ has instances $p(C) \supset [a]q(C)$ and $(\forall x\ p(x)) \supset [a](\forall x\ q(x))$. The technique of resolution with answer extraction [Green 1969b] is used to find all relevant instances of the axioms. Formulas containing free and and bound variables, as well as constant terms can be progressed and regressed.

Section 4.5.1 and 4.5.2 outlines the progression and regression algorithms respectively; numerous examples follow. Proofs of correctness appear in 4.5.4. The algorithm, unfortunately, is not guaranteed to always terminate (although it does for the examples in this thesis); 4.5.4 contains a discussion of problems that can arise. No proof of the completeness of the algorithm is attempted.

4.5.1 Progression

The progression of a state description D through an atomic action a(Z) using domain constraints G is calculated in four steps:

1. Form G(a(Z)), the set of the instances of the dynamic axioms for action a with parameter list Z.

2. Put the description D in skolem form. Use the antecedents of the wffs in G(a(Z)) to construct an existential query on the skolem form of D, Gs, and Ge.

3. Generate and conjoin all answers to the query.

4. Transform the conjunction of answers by replacing each answer-wff by a corresponding instance of a dynamic axiom consequence. Replace skolem-functions introduced in step 2 by bound variables.

The four steps are now described in detail.*

First, check whether D is inconsistent. If so, the postcondition is simply false. Otherwise, proceed as follows:

Step 1: Form G(a(Z))

Collect all the dynamic axioms containing action symbol a, including instances of schema A15 and A16, and rename all bound variables apart. Rewrite the axioms, if necessary, so that the antecedent of each dynamic axiom is a conjunction of literals, and the consequence is a disjunction of literals. For example, the single axiom

$$\forall Y((p1(y) \vee p2(y)) \supset [a(Y)]\ (q1(Y) \wedge q2(Y)))$$

can be rewritten as four axioms

$$\forall Y\ (p1(Y) \supset [a(Y)]\ q1(Y))$$

$$\forall Y\ (p2(Y) \supset [a(Y)]\ q1(Y))$$

$$\forall Y\ (p1(Y) \supset [a(Y)]\ q2(Y))$$

$$\forall Y\ (p2(Y) \supset [a(Y)]\ q2(Y))$$

Also rewrite the axioms, if necessary, to assure that all parameters of a in each axiom are distinct bound variables. For example,

$$p \supset [a(C1)]\ q$$

is equivalent to

$$\forall y((p \wedge y=C1) \supset [a(y)]\ q)$$

---

*Notation: Capital letters (such as Z) are used for lists of terms, and small letters for matrices of formulas. By p(X) we mean a quantifier-free non-modal formula whose free variables are a subset of X; by p(T), we mean a formula of the same form with terms T substituted for X.

Form G(a(Z)), the Z-instance of a, by substituting Z for the corresponding parameters in each axiom. Let F be the free variables which appear in D or Z (i.e., the plan parameters and, as described in chapter 7 the formal objects). G(a(Z)) is a set of j formulas of the form

$$(\forall Yi)(pi(Yi,F) \supset [a(Z)]\ qi(Yi,F)) \quad *$$

(We may, at this point, throw out those axioms which have inconsistent antecendents; for example, when the substitution instance of the above axiom is

$$(p \wedge C2=C1) \supset [a(C2)]\ q$$

The more general form is required when the parameters, Z, include free variables.)

Step 2: Form Query

D is E-first, and G is equivalent to an E-first formula, the conjunction of all the static axioms. So say

$$D = (E\ Xb)(\forall\ Yb)\ d(Xb,Yb,F)$$

$$Gs = (E\ Xg)(\forall\ Yg)\ g(Xg,Yg)$$

A query is an E-quantified non-modal formula. An answer is a disjunction of variants of the query (formulas like the query with different terms substituted for the E-quantified variables), which is a consequence of the axioms. We wish to find the "strongest" answer to the query

---

* That is, the free variables in the antecedent and consequence of the wff are a subset of Yi u F.

$$G \vdash D \supset ((E\ Y1)p1(Y1,F) \vee (E\ Y2)p2(Y2,F) \vee \ldots \vee (E\ Yj)pj(Yj,F))$$

--an answer from which all other possible answers can be derived. Intuitively, we want to know all the ways in which the initial world description together with the non-modal axioms satisfy the action preconditions.

Our method is to generate and conjoin all answers to the query

$$\vdash ((\forall\ Yb)d(Xb,Yb,F) \wedge (\forall\ Yg)g(Xg,Yg) \wedge Ge) \supset ((E\ Y1)p1(Y1,F) \vee \ldots \vee (E\ Yj)pj(Yj,F))$$

Note that E-quantifiers in the antecedent have been dropped. We are assuming that the dynamic axioms are not needed to answer the query.

Resolution proceeds as follows. The free variables in Xb, Xg, and F are converted to 0-ary skolem functions Xb', Xg', and F'. The base clauses will consist of the skolem forms of the initial world description, the static axioms, and Ge, the axioms for inequality between different constants.

base clauses:

d(Xb',Yb,F')

g(Xg',Yg)

Ge

The query is negated and skolemized, using the same skolem functions for the free variable in F as before. Each ¬pi(Yi,F') falls in a separate clause. Following [Green 69b], special "answer" predicates are attached to each query clause to trace the substitutions made for each variable in each Yi. The base clauses are augmented by:

query clauses:

¬p1(Y1,F') v ans1(Y1)

¬p2(Y2,F') v ans2(Y2)

. . .

¬pj(Yj,F') v ansj(Yj)

The standard resolution rules can be strengthened with rules for equality, such as paramodulation (described in [Chang 73]), so that explicit substitution axioms for equality (instances of schema A12) need not appear.

Step 3: Generate All Answers

Create a breadth-first tree of all resolutions, without ever resolving or paramodulating directly against an "ans" literal. Each answer clause is a deduced clause containing only ans literals: a disjunction of variants of some of the ansi(Yi)'s. In each variant, some list of constants, skolem functions, and variables replaces each Yi.

Resolution continues until no significant new clauses can be created, or (perhaps) some pre-established time limit is reached. (Section 4.5.3 discusses some conditions under which resolution does not naturally terminate.)

A double-subscripted "anshl(Thl')" is the l-th variant appearing in the h-th answer. Thl' is the particular variant creating substitution, and n(h) is the number of of literals which appear in that answer. For instance, the second answer is

ans21(T21') v ans22(T22') v ... v ans2n(2)(T2n(2)')

Different variants of the same ansi may appear in an answer clause. Form the conjunction of all m answers:

(ans11(T11') v ... v ans1n(1)(T1n(1)'))
^ ... ^
(ansm1(Tm1') v ... v ansmn(m)(Tmn(m)'))

Step 4: Transform Answer Formula

Corresponding to each variant anshl(Thl') is a dynamic axiom instance phl(Thl',F') ⊃ qhl(Thl',F'). Substitute for each anshl(Thl') in the conjunction of all answer clauses the corresponding qhl(Thl',F').

(q11(T11',F') v ... v q1n(1)(T1n(1)',F')
^ ... ^
(qm1(Tm1',F') v ... v qmn(m)(Tmn(m)',F') *

Let U be the list of all variables which appear in any Thl'. Let Thl be obtained from Thl' by replacing terms from Xb', Xg', or F' by the original variables from Xb, Xg, or F. The strongest provable postcondition is then

(E Xb)(E Xg)(∀ U)
((q11(T11,F) v ... v q1n(1)(T1n(1),F))
^ ... ^
(qm1(Tm1,F) v ... v qmn(m)(Tmn(m),F)))

4.5.2 Regression

The outline of the regression algorithm mirrors the progressive case. The task is to find the instances of the dynamic axioms whose consequences imply the state description. The existential query formed by the algorithm is the contrapositive of this implication:

---

* Formulas of the above form are abbreviated as (q11(T11',F') v^ etc) in the following sections.

it asks for the disjunction of negated dynamic axiom consequences which are implied by the negation of the state description.

Step 1: Form G(a(Z))

Gather the axioms for a and form the Z-instance of a as before.

Step 2: Form Query

Let D be the ∀-first goal world description and a(Z) the action. Where F is the list of free variables in D and Z,

D = (∀ Yb)(E Xb)d(Xb,Yb,F)

Gs= (E Xg)(∀ Yg)g(Xg,Yg)

The general form of the existential query is:

G⊢ ¬D ⊃ ((E Y1)¬q1(Y1,F) v
(E Y2)¬q2(Y2,F) v
. . . v
(E Yj)¬qj(Yj,F))

F', Yb', and Xg' are lists of skolem functions corresponding to variables F, Yb, Xg. Resolve the clause set:

base clauses:

¬d(Xb,Yb',F')

g(Xg',Yg)

Ge

query clauses:

q1(Y1,F') v ans1(Y1)

. . .

qj(Yj,F') v ansj(Yj)

Step 3: Generate All Answers

If an empty answer arises then it must be the case that G⊢B, and the complete precondition is simply true. Otherwise, form the negation of the conjunction of all the answers (with the same assumption that only a finite number will be generated).

¬((ans11(T11') v ... v ans1n(1)(T1n(1)))
^ ... ^
(ansm1(Tm1') v ... v ansmn(m)(Tmn(m))))

Step 4: Transform Answer Formula

Replace each variant anshl(Thl') by the corresponding ¬phl(Thl',F') and simplify.

(p11(T11',F') ^ ... ^ p1n(1)(T1n(1)',F')
v ... v
(pm1(Tm1',F') ^ ... ^ pmn(m)(Tmn(m)',F')

Let U be the list of all variables in any Thl'; replace the skolem functions in each Thl' by the original variables from Yb, Xg, and F. The weakest provable precondition is then the disjunction of conjunctions,

(∀ Yb)(∀ Xg)(E U)
((p11(T11,F) ^ ... ^ p1n(1)(T1n(1),F))
v ... v
(qm1(Tm1,F) ^ ... ^ pmn(m)(Tmn(m),F)))

henceforth abbreviated (p11(T11,F) ^v etc). The precondition, like the orignal goal state, is ∀-first.

4.5.3 Examples Of Progression And Regression

For clarity of exposition of the following examples, I assume that the answer generator is sophisticated enough not to produce redundant answers.

Disjunctive Information

Recall the propositional example from [Rosenschein 81], which uses the following axioms.

Gd

$$A \supset[a] (B \vee C)$$
$$G \supset[a] \neg B$$
$$(F \wedge E) \supset[a] D$$

In this case G(a) is simply the same as Gd.

Ex 1: Calculate $A \wedge G/a$

base clauses
- A
- G

query clauses
- $\neg A \vee ans1$
- $\neg G \vee ans2$
- $\neg \Gamma \vee \neg E \vee ans3$

answer clauses
- ans1
- ans2

postcondition
- $(B \vee C) \wedge \neg E = C$

Ex 2: Calculate $a\backslash(C \vee D)$

base clauses
- $\neg C$
- $\neg D$

query clauses
- $B \vee C \vee ans1$
- $\neg B \vee ans2$

D v ans3

answer clauses
ans1 v ans2
ans3

precondition
(A ∧ G) v (F ∧ E)

We see how v and ∧ are reversed when forming the precondition.

Quantified Information

The next series of examples come from the delightful world of electronic warfare. Interpret mx(CITY) and hq(CITY) as CITY is the location of an mx missile silo and CITY is a command headquarters respectively. Bombing a city destroys any missile silo there, but all other silos are unaffected.

G

(1) ∀y (¬mx(y) ⊃ [bomb(y)] ¬mx(y))
(2) ∀yz ((mx(z) ∧ y≠z) ⊃ [bomb(y)] mx(z))

pure frame axioms:

(3) ∀yz (¬mx(z) ⊃ [bomb(y)] ¬mx(z))
(4) ∀yz (hq(z) ⊃ [bomb(y)] hq(z))
(5) ∀yz (¬hq(z) ⊃ [bomb(y)] ¬hq(z))
(6) ∀yzw(z=w ⊃ [bomb(y)] z=w)
(7) ∀yzw(z≠w ⊃ [bomb(y)] z≠w)

Ex 3: Calculate Ex(mx(x) ∧ hq(x) ∧ mx(MIAMI))/bomb(MIAMI)

The set of formulas G(bomb(MIAMI)) is formed by substituting MIAMI for y in each of the dynamic axioms.

G(bomb(MIAMI))

mx(MIAMI) ⊃ [bomb(MIAMI)] ¬mx(MIAMI)
∀ z ((mx(z) ∧ MIAMI≠z) ⊃ [bomb(MIAMI)] mx(z))
∀ z (¬mx(z) ⊃ [bomb(MIAMI)] ¬mx(z))
∀ z (hq(z) ⊃ [bomb(MIAMI)] hq(z))
∀ z (¬hq(z) ⊃ [bomb(MIAMI)] ¬hq(z))
∀ zw (z=w ⊃ [bomb(MIAMI)] z=w)
∀ zw (z≠w ⊃ [bomb(MIAMI)] z≠w)

The general form of the query is

$$\vdash (mx(x) \wedge hq(x) \wedge mx(MIAMI)) \supset$$
$$mx(MIAMI) \vee$$
$$Ez(mx(z) \wedge z \neq MIAMI) \vee$$
$$Ez(\neg mx(z)) \vee$$
$$Ez(hq(z)) \vee$$
$$Ez(\neg hq(z)) \vee$$
$$Ezw(z=w) \vee$$
$$Ezw(z \neq w)$$

Rewriting this in clausal form:

base clauses
- mx(X)
- hq(X)
- mx(MIAMI)

query clauses
- ¬mx(MIAMI) v ans1
- ¬mx(z) v z=MIAMI v ans2(z)
- mx(z) v ans3(z)
- ¬hq(z) v ans4(z)
- hq(z) v ans5(z)
- z≠w v ans6(z,w)
- z=w v ans7(z,w)

X is a skolem function replacing the existentially quantified variable x in the world description. The (pruned) answer set contains

answer clauses
- ans1
- ans2(X) v ans6(X,MIAMI)
- ans4(X)

This corresponds to the valid formula

$$(Ex(mx(x) \wedge hq(x) \wedge mx(MIAMI))) \supset$$
$$Ex(mx(MIAMI) \wedge$$
$$((mx(x) \wedge x \neq MIAMI) \vee x=MIAMI) \wedge$$
$$hq(x))$$

which yields

postcondition
$$Ex(\neg mx(MIAMI) \wedge$$
$$(mx(x) \vee x=MIAMI) \wedge$$
$$hq(x))$$

Almost any action can alter existentially-quantified information. Before, we knew that something was both a headquarters and a missile silo; afterwards, the second condition may have been invalidated. The postcondition is equivalent to $\neg mx(MIAMI) \wedge Ex(hq(x))$.

Ex 4: Calculate bomb(NYC)\$\forall x(\neg mx(x))$

base clauses
mx(X)

query clauses
$\neg mx(NYC) \vee ans1$
$mx(z) \vee ans2(z)$
$\neg mx(z) \vee ans3(z)$
$hq(z) \vee ans4(z)$
$\neg hq(z) \vee ans5(z)$
$z=w \vee ans6(z,w)$
$z \neq w \vee ans7(z,w)$

answer clauses
ans1 v ans6(X,NYC)
ans3(X)

precondition
$\forall x((mx(NYC) \wedge x=NYC) \vee \neg mx(x))$

That is, $\forall x(mx(x) \supset x=NYC)$, which agrees with our intuitions: if no silos are left after bombing NYC, beforehand NYC could have been the only silo.

Ex 5: hit(NYC)\$Ex(\neg mx(x))$

base clauses
mx(x)

query clauses
same as previous example

answer clauses
ans1
ans3(x)

precondition

$$Ex(mx(NYC) \vee \neg mx(x)) = true$$

An important part of any implementation will clearly be a simplification mechanism. The cost of regressing $Ex(mx(NYC \vee \neg mx(x))$ through further actions would be many times the cost of simply regressing true.

Multiple Effects

A single dynamic axiom can represent an action which affects an arbitrarily large number of entities. Consider an action which bombs all cities where headquarters are located.

$$^{Gd}(1) \quad \forall y\ (hq(y) \supset [bomballhq]\ \neg mx(y))$$

frame axioms

$$(2) \quad \forall z\ ((mx(z) \wedge \neg hq(z)) \supset [bomballhq]\ mx(z))$$
$$(3) \quad \forall z\ (\neg mx(z) \supset [bomballhq]\ \neg mx(z))$$
$$(4) \quad \forall z\ (\ hq(z) \supset [bomballhq]\ hq(z))$$
$$(5) \quad \forall z\ (\neg hq(z) \supset [bomballhq]\ \neg hq(z))$$
$$(6) \quad \forall zw(\ z=w \supset [bomballhq]\ z=w)$$
$$(7) \quad \forall zw(\ z\neq w \supset [bomballhq]\ z\neq w)$$

Ex 6: Calculate $(hq(LA) \wedge hq(BUF))/bomballhq$

base clauses

$hq(LA)$

$hq(BUF)$

query clauses

$\neg hq(y) \vee ans1(y)$

$\neg mx(z) \vee ans2(z)$

$mx(z) \vee ans3(z)$

$\neg hq(z) \vee ans4(z)$

$hq(z) \vee ans5(z)$

$z\neq w \vee ans6(z,w)$

$z=w \vee ans7(z,w)$

answer clauses

ans1(LA)
ans1(BUF)
ans4(LA)
ans4(BUF)
ans1(w) v ans5(w)

postcondition
∀w (¬mx(LA) ∧ ¬mx(BUF) ∧
hq(LA) ∧ hq(BUF) ∧
(¬hq(w) v ¬mx(w)))

The postcondition simplifies to

∀w (hq(w) ⊃ ¬mx(w)) ∧ hq(LA) ∧ hq(BUF)

Multiple Axioms

A final example demonstrates the step in which axioms with disjunctive preconditions or conjunctive postconditions are rewritten as sets of axioms. Say G contains

∀xy (p(x,y) ⊃ [a] (q(x) ∧ t(y)))

Rewrite this in G(a) as

∀ x1 y1 (p(x1,y1) ⊃ [a] q(x1))

∀ x2 y2 (p(x2,y2) ⊃ [a] t(x2))

One may confirm that

a\(q(C) ∧ t(D)) = (E x2 y1)(p(C,y1) ∧ p(x2,D))

which is a precondition weaker than the more obvious p(C,D).

4.5.4 Correctness Of Progression And Regression

We demonstrate the correctness of the progression and regression algorithms described in 4.5.1 and 4.5.2.

4.5.4.1 Correctness Of Progression -

We show that the implementation of "/" is correct: that $G\vdash D \supset [a(Z)]\ D/a(Z)$.

The descriptive formula D can become false when conditionals are introduced in the presence of multiple constraints. Since

$\vdash$ false $\supset$ [a(Z)] p

$\vdash$ false $\supset$ [a(Z)] $\neg$p

then false/a(Z) $\supset$ false.

Axioms A3 and A4 and rule R2 allow renaming of bound variables in modal formula, as well as the substitution of particular terms for parameters when Ga is formed. The correctness demonstration proceeds in two stages. First, we show that

$G\vdash D \supset (E\ Xb)(E\ Xg)(\forall\ U)(p11(T11,F)\ v\wedge\ etc)$

Second, that

$G\vdash (E\ Xb)(E\ Xg)(\forall\ U)(p11(T11,F)\ v\wedge\ etc) \supset$
$[a(Z)]\ (F\ Xb)(E\ Xg)(\forall\ U)(q11(T11,F)\ v\wedge\ etc)$

Modus ponens completes the proof.

Lemma 1: $G\vdash D \supset (E\ Xb)(F\ Xg)(\forall\ U)(p11(T11,F)\ v\wedge\ etc)$

We know that the use of tracing functions is a correct method for implementing answer extraction. Therefore the formulas

ph1(Th1',F') v ... v phn(h)(Thn(h)',F')

are deducible from the bases clauses for 1<=h<=m. By the deduction principle of first-order logic,

$((\forall\ YbYg)(d(Xb',Yb,F') \wedge g(\lambda g',Yg) \wedge Ge)) \supset$
$(\forall\ U)(ph1(Th1',F')\ v\ \ldots\ v\ phn(h)(Thn(h)',F'))$

is a tautology for 1<=h<=m. Therefore

((∀ YbYg)(d(Xb',Yb,F') ∧ g(Xg',Yg) ∧ Ge)) ⊃
(∀ U)(p11(T11',F') v∧ etc)

is valid. One can substitute a free variable for a 0-ary function in a tautology and still have a tautology. So the skolem functions can be eliminated:

((∀ YbYg)(d(Xb,Yb,F) ∧ g(Xg,Yg) ∧ Ge)) ⊃
(∀ U)(p11(T11,F) v∧ etc)

Since this is a valid formula of first-order logic expressible in TDL, TH1 tells us that this is a theorem of TDL. Simple first-order manipulations further transform the formula.

⊢ ((E XbXg)(∀ YbYg)(d(Xb,Yb,F) ∧ g(Xg,Yg) ∧ Ge)) ⊃
(E XbXg)(∀ U)(p11(T11,F) v∧ etc)

⊢ ((E Xg)(∀ Yg)g(Xg,Yg) ∧ Ge) ⊃
(((E Xb)(∀ Yb)d(Xb,Yb,F)) ⊃
(E XbXg)(∀ U)(p11(T11,F) v∧ etc))

that is

⊢ (Gs ∧ Ge) ⊃ (D ⊃ (E XbXg)(∀ U)(p11(T11,F) v∧ etc))

By TH2,

G⊢ D ⊃ (E XbXg)(∀ U)(p11(T11,F) v∧ etc)

Lemma 2: G⊢ (E Xb)(E Xg)(∀ U)(p11(T11,F) v∧ etc) ⊃
[a(Z)] (E Xb)(E Xg)(∀ U)(q11(T11,F) v∧ etc)

The second part is also straightforward. By A3 derive from Gd the series of formulas

G⊢ p11(T11,F) ⊃ [a(Z)] q11(T11,F)

. . .

G⊢ pmn(m)(Tmn(m),F) ⊃ [a(Z)] qmn(m)(Tmn(m),F)

Propositional manipulations combine the above.

G⊢ (p11(T11,F) v∧ etc) ⊃
(([a(Z)]q11(T11,F) v ... v [a(Z)]q1n(1)(T1n(1),F))
∧ ... ∧
([a(Z)]qm1(Tm1,F) v ... v [a(Z)]qmn(m)(Tmn(m),F)))

TH5 and TH6 permit simplification of the consequence.

$$G\vdash (p11(T11,F) \vee\wedge \text{etc}) \supset [a(Z)](q11(T11,F) \vee\wedge \text{etc})$$

Insert quantifiers using DR1 and DR2.

$$G\vdash ((E\ XbXg)(\forall\ U)(p11(T11,F) \vee\wedge \text{etc})) \supset (E\ XbXg)(\forall\ U)[a(Z)](q11(T11,F) \vee\wedge \text{etc})$$

TH4 lets us push the quantifiers through the action.

$$G\vdash ((E\ XbXg)(\forall\ U)(p11(T11,F) \vee\wedge \text{etc})) \supset [a(Z)]\ (E\ XbXg)(\forall\ U)(q11(T11,F \vee\wedge \text{etc})$$

The final result is that the calculated postcondition does indeed hold after the action:

$$G\vdash D \supset [a(Z)]\ (E\ Xb)(E\ Xg)(E\ U)(q11(T11,F) \vee\wedge \text{etc})$$

The algorithm is correct.

4.5.4.2 Correctness Of Regression -

The steps are now

$$G\vdash ((\forall\ YbXg)(E\ U)(q11(T11,F) \wedge\vee \text{etc})) \supset D$$

and

$$G\vdash ((\forall\ YbXg)(E\ U)(p11(T11,F) \wedge\vee \text{etc})) \supset [a(Z)]\ (\forall\ YbXg)(E\ U)(q11(T11,F) \wedge\vee \text{etc})$$

From the argument made earlier for the correctness of answer extraction and the removal of skolem functions,

$$\vdash (\forall\ Xb)(\forall\ Yg)(\neg d(Xb,Yb,F) \wedge g(X_b,Yg) \wedge Ge) \supset (\forall\ U)((\neg q11(T11,F) \vee \ldots \vee \neg q1n(1)(T1n(1),F)) \wedge \ldots \wedge (\neg qm1(Tm1,F) \vee \ldots \vee \neg qmn(m)(Tmn(m),F))))$$

hence

$$\vdash ((E\ X_b)(\forall\ Yg)g(Xg,Yg) \wedge Ge) \supset (((E\ Yb)(\forall\ Xb)\neg d(Xb,Yb,F)) \supset (E\ YbXg)(\forall\ U)(\neg q11(T11,F) \vee\wedge \text{etc}))$$

$$\vdash (Gs \wedge Ge) \supset (((\forall\ YbXg)(E\ U)(q11(T11) \wedge v\ etc)) \supset (\forall\ Yb)(E\ Xb)d(Xb,Yb,F))$$

So

$$G\vdash ((\forall\ YbXg)(E\ U)(q11(T11,F) \wedge v\ etc)) \supset (\forall\ Yb)(E\ Xb)d(Xb,Yb,F)$$

The sketch of the proof of the second step is the same as for progression. Modus ponens then gives correctness of regression:

$$G\vdash ((\forall\ Yb)(\forall\ Xg)(\forall\ U)(p11(T11,F) \wedge v\ etc)) \supset [a(Z)]\ D$$

4.5.5 Termination Of The Progression/Regression Algorithms

The answer-generation process always terminates in the propositional case because only a finite number of clauses can be formed from a finite set of propositions (0-ary predicates). Although resolution (or any other complete question-answering mechanism) does not always terminate for full first-order logic, one would hope that the quantifier-ordering rules suffice to guarantee termination.

Unfortunately, this is not always the case. Although the number of ground instances of all the clauses is finite, situations arise in which arbitrarily long clauses containing variables are created. This phenomenon is a concern to researchers in deductive question-answering on data bases. Infinite deductions are possible

when the set of clauses contains cycles [Lewis 75]*; a cycle arises, for example, from the recursive axiom

$\forall xyz\ ((on(x,y) \wedge above(y,z)) \supset above(x,z))$

Even if no cycles appear in the clause form of Gs and the world description D, cycles may appear when the query clauses are added.

Several situations can arise in the answer-generation portion of the algorithm:

1. A finite number of answers are generated and the algorithm halts.

2. Only a finite number of answers need be generated, but some cycle(s) lead to an infinite series of resolutions.

3. An infinite number of significantly different answers are generated. The strongest provable postcondition (weakest provable precondition) cannot be represented by a finite E-first ($\forall$-first) formula.

---

* Formally, a cycle is a set of clauses of the form

$$\neg L1(X1) \vee L2'(X2') \vee \ldots$$
$$\neg L2(X2) \vee L3'(X3') \vee \ldots$$
$$\neg L3(X3) \vee L4'(X4') \vee \ldots$$
$$\vdots$$
$$\neg Ln(Xn) \vee L1'(X1') \vee \ldots$$

where $Li(Xi)$, $Li'(Xi')$ are literals, and there is a unifying substitution T such that $Li(T)=Li'(T)$ for $1 \leq i \leq n$.

[Reiter 78] discusses a number of techniques for "neutralizing" certain forms of cycles.

The examples presented in this paper fall in the first two categories. In these examples, furthermore, the simple heuristic of subsumption cuts all infinite deductive paths [Chang 73]. A clause L subsumes a clause M if a variant of L is contained in M. We eliminate subsumed clauses, as they are generated, from the clause set. The ans predicates are considered part of the clause when testing for subsumption. E.g:

p(x) v ans1(x) does subsume p(y) v ans1(C44)

p(x) v ans1(x) does not subsume p(C1)

The notion is that a subsuming clause must not have an ancestor derived from a dynamic axiom, which is lacked by the subsumed clause. For each answer clause lost because one of its ancestors is subsumed, there is an equivalent or stronger answer clause with the subsuming clause as ancestor.

The following planning problem demonstrates the third case, where heuristics are to no avail, because the regression of a formula is not finitely realizable. Let G contain only

Gs

$\forall xyz\ ((on(x,y) \wedge above(y,z)) \supset above(x,z))$

$\forall xy\ (on(x,y) \supset above(x,y))$

Gd

$\forall xy\ (on(x,y) \supset [A]\ on(x,y))$

Try calculating A\above(C,D). The weakest computable precondition is

$on(C,D) \vee$
$(\exists\ x1)(on(C,x1) \wedge on(x1,D)) \vee$
$(\exists\ x1x2)(on(C,x1) \wedge on(x1,x2) \wedge on(x2,D)) \vee$
$(\exists\ x1x2x3)(on(C,x1) \wedge on(x1,x2) \wedge on(x2,x3) \wedge on(x3,D))$
$\vee$ ... forever

The obvious precondition, above(C,D), can't be obtained, because we have deliberately neglected to provide a frame axiom for above.

Simply stopping the answer generation process at some pre-set time limit, or establishing a length-limit for clauses, does not affect correctness (but of course risks the loss of some solutions). The ultimate solution may be to revamp BIGRESS so that it does not calculate the entire progression or regression of a formula at once, but only portions on an "as needed" basis.*

4.5.6 Efficiency Of Regression And Progression

We have begged the question of efficiency; having left the warm confines of STRIPS, the combinatorial problems of frame axioms appear in full force.

In particular, the interaction of the frame axioms for equality and the theorem-prover's mechanism for handling equality can cause combinatorial explosion intolerable in even the most trivial implementation. The clause "x=y" paramodulates against <u>any</u> other clause. Yet no significant answers are lost if the theorem prover only resolves this clause so that at least one of its variables unifies with a skolem function. Equality predicates in clauses should be evaluated whenever possible, and tautologies removed.

The postcondition is not weakened (precondition is not strengthened) if ground instances of invariant predicates which

---

* See [Waldinger 77] for an argument for the undesirability of calculating, at once, the complete progression of a condition through an action.

appear as domain constraints are disregarded by the theorem prover. The frame axioms for such predicates force the resolution procedure to extract the "intensions" of certain answer sets. This means, in particular, that not all of Ge need be worked with.

For example, suppose

Gs

virus(GERM1)

virus(GERM2)

. . .

virus(GERM99999)

Gd

∀y(virus(y) ⊃[irradiate] mutated(y))

∀y(virus(y) ⊃[irradiate] virus(y))

∀y(¬virus(y) ⊃[irradiate] ¬virus(y))

Virus is invariant. Using the full set of axioms in calculating the effect of irradiate on any state gives

true/irradiate = ∀y(¬virus(y) v mutated(y))
∧ mutated(GERM1)
∧ mutated(GERM2)
∧ . . .
∧ mutated(GERM99999)

If no axioms from Gs are used, one obtains the shorter

true/irradiate = ∀y(¬virus(y) v mutated(y))

These formulas are equivalent under G.

Every significant answer should have an ancestor in either a non-frame dynamic axiom or the given world description. Completeness is not weakened, since the only answers lost lead to post/pre conditions derivable from the static axioms alone. This may be the key to a successful implementation: it helps keep keeps

the information stored in the static axioms from being duplicated in each world description.

The very lack of a built-in solution (with corresponding built-in limitations) to the frame and equality problems in the DL approach makes it a good framework for testing various solvents against these viscous difficulties.

4.6 DETERMINISTIC CONTROL STRATEGIES

The bare-bones BIGRESS algorithm says little about how actions and tests are actually chosen, and at what point backtracking occurs. To implement BIGRESS on a real, serial machine, its non-deterministic "choice" operation must be simulated by a deterministic control strategy. Such a strategy seeks to strike a balance between two goals: finding any possible solution (completeness), and finding a solution quickly (efficiency).

As shown in 4.3.2, "non-deterministic" BIGRESS's completeness rests on completeness of the progression and regression functions. When the space of state descriptions is finite (as in the propositional case), a depth-first deterministic strategy can be complete, given completeness of the auxiliary functions. Loops in the search space can be eliminated by extra checking in Liveforward and Livebackward, as suggested in [Rosenschein 81]. Let BIGRESS take as parameters not only the lists R and S of the current intial and goal descriptions, but descriptions of all previously visited states. Prune progressed descriptions implied by any old description, or regressed descriptions which imply any old

description.

In the first-order case, the state space can be infinite, as in the second socks example (section 5.2). Some form of breadth-first search is needed for completeness. For example: Effectively enumerate the sets of atomic actions and non-modal wffs. Initialize some global variable I to 1. Find all plans of length less than or equal to I (i.e., I calls to BIGRESS) containing no action or test numbered higher than I. Increment I and repeat.

A breadth-first bidirectional search (alternating choices from Liveforward and Livebackward) is suprisingly efficient for fairly short plans [Nilsson 80]. Efficiency is increased by heuristically ordering the sets generated by Liveforward, Livebackward, and Nontriv. As in STRIPS, one could analyze the partial proof constructed in a failed stopping test as a basis for guessing a good next action.

* * *

In summary: we defined a first-order planning problem, and extended the propositional provability, progression, and regression functions. Correctness was proved, and problems of completeness discussed.

CHAPTER 5

EXAMPLES OF PLANNING IN TDL

I shall present a number of planning problems which demonstrate the strengths of the TDL framework. Examples shall include disjunctive goals and world descriptions, derived predicates, actions with side effects, nondeterministic actions, and "distributed" action definitions; all features which are difficult to accomcdate in other systems.

5.1 BLOCKS AND BOXES WORLD

That hoariest of settings, the blocks world, can still provide some challenging problems. The domain will contain a table and a number of boxes and blocks. A block can be supported by the table or one of the boxes. (Blocks won't be stacked directly on other blocks.) Blocks can be cubes, cones, and spheres. The static axioms include:

box(BOX1), box(BOX2), box(BOX3)
block(BLK1), ..., block(BLK9)
∀x ((box(x) ∨ x=TABLE) ≡ sup(x))
∀xyz ((on(x,y) ∧ on(x,z)) ⊃ y=z)
∀xyz ((shape(x,y) ∧ shape(x,z)) ⊃ y=z)

Domain constraints shall be added as needed to illustrate various features. Assume that Gd includes appropriate frame axioms for all actions so that box, block, and shape are invariant.

Suppose that a robot hand is able to manipulate one block at a time.

$\forall$ xy ((on(x,y) $\wedge$ handempty) $\supset$ [pickup(x,y)] hold(x))

$\forall$ xy ((hold(x) $\wedge$ sup(y)) $\supset$ [putdown(x,y)] (on(x,y) $\wedge$ handempty))

frame axioms:

$\forall$ xyzw (on(z,w) $\wedge$ z$\neq$x) $\supset$ [pickup(x,y)] on(z,w))

$\forall$ xyzw (on(z,w) $\supset$ [putdown(x,y)] on(z,w))

A plan constraint can include an indefinite goal; for instance, to obtain a cube or a cone on BOX2 from a state where

shape(BLK3,CUBE)
shape(BLK4,SPHERE)
shape(BLK5,CONE)
on(BLK3,BOX1)
on(BLK4,TABLE)
on(BLK5,TABLE)

is true, Gs should be expanded to include the first three propositions above, and RES(u) should be

(on(BLK3,BOX1) $\wedge$ on(BLK4,TABLE) $\wedge$
on(BLK5,TABLE) $\wedge$ handempty)
$\supset$ [u] (E x)(on(x,BOX2) $\wedge$ (shape(x,CUBE)
$\vee$ shape(x,CONE))))

Of course, BIGRESS can return either

[pickup(BLK3,BOX1);putdown(BLK3,BOX2)]

or

[pickup(BLK5,TABLE);putdown(BLK5,BOX2)]

A more interesting problem arises when less is known about the shape of the blocks. Use the same plan constraint when Gs contains only

shape(BLK3,CUBE) v shape(BLK5,CUBE)

A straightforward search by the planner could come up with the solution

[pickup(BLK3,BOX1);putdown(BLK3,BOX2);
pickup(BLK5,TABLE);putdown(BLK5,BOX2)]

Since one of BLK3 or BLK5 must be a cube, putting both on BOX2 assures that at least a cube is there. If the predicate shape can be used as a test, then a more efficient solution can be found.

[shape(BLK3,CUBE) →
(pickup(BLK3,BOX1);putdown(BLK3,BOX2)),
(pickup(BLK5,TABLE);putdown(BLK5,BOX2))]

Note that NonTriv would never return the test shape(BLK4,CUBE), because one can derive from the axioms ¬shape(BLK4,CUBE).

The blocks world I've described so far lacks any interesting interactions between successive actions. To liven things up a bit, the boxes will be given lids which can be opened or closed. Blocks can be put in or removed from a box when the box is open. When the lid is closed, blocks can be piled on top of the box. Let Gd contain

∀ xy ((in(x,y) ∧ ¬closed(y) ∧ handempty) ⊃
[pickup(x,y)] hold(x))

∀ xy ((on(x,y) ∧ handempty ⊃
[pickup(x,y)] hold(x))

∀ xy ((hold(x) ∧ box(y) ∧ ¬closed(y)) ⊃
[putdown(x,y)] (in(x,y) ∧ handempty))

∀ xy ((hold(x) ∧ sup(y) ∧ closed(y)) ⊃
[putdown(x,y)] (on(x,y) ∧ handempty))

∀ x ((¬closed(x) ∧ box(x)) ⊃
[shut(x)] closed(x))

∀ x ((closed(x) ∧ box(x)) ⊃
[open(x)] ¬closed(x))

Note that the effect of the putdown action depends upon the state of

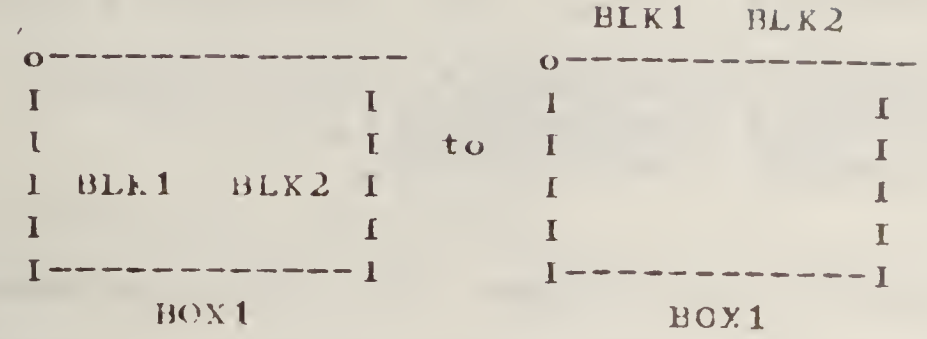
world in which it is performed: if the lid is open, the block goes into the box; otherwise, it goes onto the box. Gs should include closed(TABLE). The frame axioms are straightforward, except for the open action. It has the side effect of dumping all the blocks which were resting on the lid onto the table. The frame axioms are:

```
∀ xwz  ((on(w,z) ∧ z≠x) ⊃
              [open(x)] on(w,z))

∀ xw   ((closed(x) ∧ box(x) ∧ on(w,x)) ⊃
              [open(x)] on(w,TABLE))

∀ xywz ((in(w,z) ∧ w≠x) ⊃
              [pickup(x,y)] in(w,z))

∀ xywz ((on(w,z) ∧ w≠x) ⊃
              [pickup(x,y)] on(w,z))

+ pure frame axioms for
      open:    hold, in
      shut:    hold, in, on
      pickup:  closed
      putdown: closed, in, on
```

Open is thus an influential action, as discussed in chapter 2: it affects the truth values of an arbitrary number of relationships.

"Non-linear" planners such as NOAH create separate subplans to achieve each conjunct in a conjunctive goal, then heuristically interleave the subplans to achieve the combined goal. The greater the number of potential interactions between operators -- the more ways in which one action can undo the effect of another -- the more costly a non-linear approach to planning becomes, and the more attractive a straightforward linear planner like BIGRESS. Consider the problem specified by the constraint

```
(in(BLK1,BOX1) ∧ in(BLK2,BOX1)
 ∧ closed(BOX1) ∧ handempty)
   ⊃ [u] (on(BLK1,BOX1) ∧ on(BLK2,BOX1))
```



The shortest plan to do this takes eight steps:

```
[open(BOX1);pickup(BLK1,BOX1);putdown(BLK1,TABLE);
 pickup(BLK2,BOX1);close(BOX1);putdown(BLK2,BOX1);
 pickup(BOX1,TABLE);putdown(BLK1,BOX1)]
```

A non-linear planner would create separate subplans to put BLK1 and BLK2 on BOX1.

```
[open(BOX1);pickup(BLK1,BOX1);close(BOX1);
      putdown(BLK1,BOX1)]
```

and

```
[open(BOX1);pickup(BLK2,BOX1);close(BOX1);
      putdown(BLK2,BOX1)]
```

There is no way to linearize these subplans to achieve the desired goal. [Sacerdoti 77] refers to such a situation as a "double-cross". Rather sophisticated heuristics (which are only partially implemented in NOAH) are needed to insert new steps into the plan to temporarily place BLK1 on the table while removing BLK2. BIGRESS, of course, would have to perform a great deal more pure search to find its solution; but the use of a planning hierarchy (see section 6.2) can drastically reduce the search while retaining linearity.

It is often useful to include axioms which essentially define a new predicate in terms of more basic predicates. For instance, the predicate "at" could hold of a block which is either on or in a box, and "nextto" of blocks which are on the same box:

$$\forall\, xy\ (at(x,y) \equiv (on(x,y) \vee in(x,y)))$$

$$\forall\, xy\ (nextto(x,y) \equiv ((on(x,BOX1) \wedge on(y,BOX1)) \vee (on(x,BOX2) \wedge on(y,BOX2)) \vee (on(x,BOX3) \wedge on(y,BOX3))))$$

Planning systems by [Fahlman 74] and [Fikes 75] distinguished such derived, or secondary relationships from the more basic primary relationships. Their systems gained efficiency by not explicity including rules for regressing or progressing secondary relationships through actions. The idea was that the information encoded by a secondary proposition in a world description was purely redundant. The planner would keep track of the primary relationships used in the deduction of each secondary relationship. If any of the former were falsified, the latter was deleted.

In the TDL framework, no special dynamic axioms (including frame axioms) need be included for predicates whose instances are always equivalent to a (particular) formula not containing that predicate. The planner need not keep track of how any predicate was derived; the progression or regression algorithms effectively translate secondary relationships to a primary form as needed. For instance, using the axioms described so far, the planner can calculate

$$at(BLK1,BOX1)/open(BOX1) = \neg closed(BOX1) \wedge (in(BLK1,BOX1) \vee on(BLK1,TABLE))$$

The Fahlman and Fikes systems didn't allow secondary relationships to explicitly appear in the problem description; such relationships could never be deleted, since no chain of deductions from primary relationships would be known. But in our purely

logical framework, the constraint

$$(at(BLK1,BOX2) \wedge nextto(BLK3,BLK1)) \supset [u]\neg nextto(BLK3,BLK1)$$

is perfectly acceptable.

5.2 THE SOCKS WORLD

A tire on my automobile bursts as I speed down the New York Thruway; the spare in the trunk is patched and pressurized, since I have planned for this contingency. Two simultaneous blowouts leave me stranded in the deepening snow.... You pluck a Christmas candy from the gaily wrapped box. Nougat, carmel, or cherry would satisfy your craving, but a sulphur-filled center leaves you understandably surprised and angry.

Chance plays a significant role in most everyday affairs. We are able to create and carry out plans because we assume (justifiably or not) that the random nature of events is constrained in some known way.

In the typical blocks-world scenario, chance is completely eliminated. Every action modifies the world in some known, fully-specified manner -- falsifying some propositions, establishing others. Indeed, only such actions can be represented by add/delete list pairs.

The TDL framework allows one to model non-deterministic actions. Semantically, a non-deterministic action A relates some world to more than one world:

E I,J,K such that (I,J) ϵ M(A),
(I,K) ϵ M(A), J $\neq$ K

The postcondition of a non-deterministic action and a state will generally include disjunctive and existentially-quantified information.

The famous "three-socks problem" incorporates a carefully controlled element of chance. You stand in a totally dark room before a dresser full of loose socks, some black and some white. The only action you can perform is to pull a sock from the dresser and drop it on the table. How many socks must you remove before you can be sure that a matching pair is laid out? In the following simple formulation, the predicate was_ontable holds of socks which were on the table prior to the most recent "pick" action. The proposition P states that a black or white sock is on the table which wasn't there before the most recent action.

Gs: P $\supset$ Ex(ontable(x) $\wedge$ $\neg$was_ontable(x) $\wedge$
(black(x) v white(x)))

Gd: true $\supset$ [pick] P

$\forall$y(ontable(y) $\supset$ [pick] ontable(y) $\wedge$ was_ontable(y))

frame axioms:

$\forall$y(white(y) $\supset$ [pick] white(y))

$\forall$y(black(y) $\supset$ [pick] black(y))

R(u): true $\supset$ [u] Exy (ontable(x) $\wedge$ ontable(y) $\wedge$
x$\neq$y $\wedge$
(((black(x) $\wedge$ black(y)) v
(white(x) $\wedge$ white(y)))))

BIGRESS quickly finds the solution pick;pick;pick. A forward search develops the following sequence of state descriptions.

```
true/pick = P

(true/pick)/pick =
   Fx (ontable(x) ∧ was_ontable(x) ∧
            (black(x) v white(x)) ∧
       P)

((true/pick)/pick)/pick =
   Ex x' (ontable(x) ∧ was_ontable(x) ∧
            (black(x) v white(x)) ∧
          x≠x' ∧
          ontable(x') ∧ was_ontable(x') ∧
            (black(x') v white(x')) ∧
          P)
```

The final formula implies the desired goal. Note that the planner infers (in the progression progress) that the two socks that are on the table after the second pick aren't equal, since was_ontable holds for one and ¬was_ontable for the other, and passes this inequality through the final pick action.

While this problem may not appear too impressive (no intelligence is required in the search -- only one path is possible), it is beyond the capabilities of many planning systems. Of the STRIPS-based planners discussed here, only NOAH can be programmed for actions like "pick up a random object from a pile". But NOAH's non-linear planning strategy would get it into trouble. It would deal the disjunctive goal "obtain two black OR two white socks" by creating two subproblems, one to obtain two black socks and one to obtain two white socks, with the hope of later incorporating in the final plan the subproblem solution which proved easiest to obtain. In this case, both attempts are bound to fail.

An odd feature of the above set of axioms is that although they sufficed to solve the problem, they do not guarantee the completeness of BIGRESS's search. The formula

(((true/pick)/pick)/pick) implies that the domain contains at least three entities. The frame axioms for equality and the "halting axiom" (<pick>true) allow one to conclude that

$G\vdash \exists xyz\,(x\neq y \wedge y\neq z \wedge z\neq x)$

--which can't be obtained from the static axioms alone. In fact, such reasoning reveals that the domain must be infinite in any structure satisfying the dynamic axioms.

The problem is that the socks are appearing out of thin air. A more elaborate formulation makes sure that socks come from the dresser.

Gs: $P \supset \exists x\,(ontable(x) \wedge \neg was\_ontable(x) \wedge was\_indresser(x))$

Gd: $\forall y(indresser(y) \supset [pick]\,P)$

$\forall y(ontable(y) \supset [pick]\,(was\_ontable(y) \wedge ontable(y)))$

$\forall y(indresser(y) \supset [pick]\,(was\_indresser(y) \wedge (indresser(y) \vee ontable(y))))$

$\forall y(\neg indresser(y) \supset (\neg indresser(y) \wedge \neg was\_indresser(y)))$

The most popular version of the socks problem adds the fact that the dresser initially contains exactly 4 (or any fixed number) black socks and 4 white socks. One formulation includes a domain closure axiom * for things in the dresser.

Add to Gs:

$black(S1),\ black(S2),\ black(S3),\ black(S4)$

$white(S5),\ white(S6),\ white(S7),\ white(S8)$

* $\forall y(indresser(y) \supset (y=S1 \vee \ldots \vee y=S4))$

$$k(u): \quad indresser(S1) \wedge \ldots \wedge indresser(S8) \supset [u]\, Exy\, (ontable(x) \wedge ontable(y) \wedge x \neq y \wedge ((black(x) \wedge black(y)) \vee (white(x) \wedge white(y))))$$

Any number of variations can be devised: get a pair of black socks only (which now has the solution pick;pick;pick;pick;pick;pick;pick;pick); get a particular sock; allow multiple colors, etc.

Even putting the domain closure axiom aside, it is no longer derivable that the domain is infinite... but the search space for a particular problem may remain infinite. Note that (without *)

$$true/pick \supset Ex\forall y\, (\neg indresser(y) \vee ontable(x))$$

$$(true/pick)/pick \supset Exx'\forall y\, (\neg indresser(y) \vee (ontable(x) \wedge ontable(x') \wedge x \neq x'))$$

and so forth. Starting from a state of complete ignorance (true), applying pick N times results in the knowledge that either the dresser is empty ($\forall y(\neg indresser(y))$) or N socks are on the table. It is never the case that

$$G \vdash true/pick^{**}M \supset true/pick^{**}N$$

for M<N, so the simple heuristics in LiveForward and LiveBackward to detect loops will never prune this branch of the search tree.

5.3 DUNGEONS WORLD

A particular representation of an action may fail to capture all relevant effects of that action because the representation language lacks the necessary expressive power. As we have noted, it is difficult or impossible to adequately model a non-deterministic

action if effects are specified merely by add/delete lists of literals. Yet however rich the basic framework is -- whether one uses logical axioms, or syntactic operators, or procedures -- the system designer may simply be unaware of all the relevant effects of the action. It is desirable that the action representation can be easily strengthened, and that no "incorrect" plans (incorrect in the sense of not working as expected when actually carried out in the real world) were generated using the weaker representation.

Each TDL dynamic axiom specifies some <u>partial</u> effects of an action. Additional axioms refering to the same action can be added as needed; indeed, such a modular style is encouraged. Representing an action by a number of short axioms enhances the perspicuity of the system. It appears rather more difficult to extend a planning system by, say, rewriting QLISP procedures or STRIPS' operators. Both approaches suffer from the "one operator per action" syndrome.

More importantly, TDL's cautious approach to the frame problem -- insisting on explicit frame axioms -- assures us that if

$G \vdash R \supset [Plan]S$

and G' is a consistent extension of G, that

$G' \vdash R \supset [Plan]S$

So it is always safe to use a smaller set of rules. Under the STRIPS assumption, redefining an action A by lengthening its precondition or delete list to include a proposition B insures that there will be plans which are solutions to problems using the old definition of A, but are not solutions using the new definition. For a trivial example, when

```
oldA:  pre:  P
       del:  P
       add:  Q
```

and

```
newA:  pre:  P
       del:  P, B
       add:  Q
```

then oldA is a plan to obtain $Q \wedge B$ from $P \wedge B$, but newA isn't. Even if the system designer initially suspected that A may falsify B, she would not have been justified in using the form newA in the first place; because newA is also a plan to obtain $\neg B$ from P.

It is useful to have multiple axioms for an action when it effect crucially depends upon the state of the world in which it is executed.

In the game of Dungeons and Dragons, the walk action moves a player through a maze of rooms and tunnels. The only parameter to walk is a compass direction. The planner may have a partial image of the action:

```
at(FOREST) ⊃ [walk(NORTH)] at(CAVE)

at(CAVE)   ⊃ [walk(NORTH)] at(BOTTOMLESS_PIT) v
                           at(TREASURE_ROOM)
```

Discovering that walking SOUTH from CAVE takes you to WIZARDS_LAIR could be accommodated by the addition of another axiom. If the axioms above are strengthened by adding:

```
at(CAVE) ⊃ [walk(NORTH)] at(TREASURE_ROOM)
```

the progression algorithm correctly computes that at(CAVE)/walk(NORTH) = at(TREASURE_ROOM). A "smart" planner might notice that the second axiom becomes redundant, but this is not necessary to insure correctness.

CHAPTER 6

HIERARCHICAL PLANNING

"Hierarchical decomposition" is a buzzword in almost any area of computer science, from programming languages to machine vision. The greatest appeal of the dynamic logic framework may be its formalization of a planning hierarchy.

As in the propositinal case, the dynamic axioms at one level become plan constraints at a lower level. But the presence of parameterized actions make it convenient to associate the tree structure with a solved problem.

Recall how G(a(Z)), the Z-instance of a, was formed by gathering the dynamic axioms for a and applying specialization to obtain the instances of the axioms with parameter list Z. Each node of a problem/solution tree is single-level planning problem and solution pair, in the same vocabulary. The node has a child for each instance of an action (i.e., each atomic action) which appears in its solution. Where

$$\langle\langle VOC_k , G_k , RES(u(H))\rangle, B_k \rangle$$

is a k-level node, then for each atomic action a(Z) which appears in

Bk,

$$\langle\langle VOC_{k+1}, G_{k+1}, G_k(a(Z))\rangle, B_{k+1}\rangle$$

is a child. The vocabulary (way the world is described) may change from level to level.

The tree may be constructed top-down: determine the top-node problem and solution; form a level 2 planning problem for each atomic action which appears in the top-level solution; from each level 2 problem and solution create a child node; and continue on down the tree. So, if B = a(C);a(D), then two level two planning problems are proposed; one with constraint G (a(C)), the other G (a(D)).

Alternatively, it may be possible to precompute parts of the tree. The original dynamic axioms for a, instead of a set of their instances, may simply be taken as plan constraints. The result is a generalized lower-level plan to perform the action; plans to perform instances of the action are produced by a simple syntactic transformation of the general solution. This is made precise by the following

> Theorem: Suppose <VOC, G, G'(a(Z))> has solution B.
> Then for any x1,...,xj ∈ VAR and t1,...,tj ∈ TERMS
> <VOC, G, G'(a(Z{t1/x1, ..., tj/xj}))>
> has solution B{t1/x1, ..., tj/xj}.
> Proof: Induction on B and j, repeatedly applying A3.

Two examples follow, demonstrating both approaches. The overall solution to a hierarchical planning problem is obtain from the tree by replacing each atomic action a(Z) in B by the plan b in the child with constraint G (a(Z)) in bottom-up order. The new plan at the root solves the high-level problem in the lowest-level

vocabulary.

6.1 DUNGEONS WORLD, CONTINUED

This example demonstrates a tri-level decomposition of the dungeons world mentioned in 5.3. The top level reveals rooms which contain objects, and includes an action for transfering objects from room to room. The second level introduces a robot which actually moves the objects about. Level 3 breaks the robot's motion down into unit steps. The link between levels 1 and 2 can be entirely precomputed, but that from 2 to 3 is better computed as needed.

Define the top level vocabulary and domain constraints as follows.

Level 1

VOC1

CON1: SWORD, GOLD, ..., JEWELS
HOME, CAVE, ..., LAIR

PRED1: lloc, room

ACT1: transfer

Gs1:
room(HOME), ..., room(LAIR)

Gd1:
$\forall$ tobj tstrt tfin ((lloc(tobj,tstrt) $\wedge$ room(tfin))
$\supset$ [transfer(tobj, tstrt, tfin)] lloc(tobj,tfin))

frame axiom:

$\forall$ tobj tstrt tfin tfobj tfrm
((lloc(tfobj,tfrm) $\wedge$ tfobj$\neq$tobj)
$\supset$ [transfer(tobj, tstrt, tfin)] lloc(tfobj, tfrm))

The predicate lloc holds of an object and the room in which it is located. The planning problem

<VOC1, G1, RES1(u) =
{ (loc(GOLD,CAVE) $\wedge$ loc(JEWELS,LAIR))

$\supset$ [u] (loc(GOLD,HOME) $\wedge$ loc(JEWELS,HOME)) } >

is solved by [transfer(GOLD,CAVE,HOME);transfer(JEWELS,LAIR,HOME)]. Of course, such a plan is easier synthesized than executed -- someone must go out and fetch the valuables.

Level 2 introduces a robot who can be "at" various locations. The "go" action moves him from room to room, moving as well any objects he is carrying. Grab and drop, as their names suggest, initiate and terminate the carrying of an object.

The "basic" level 2 vocabulary and axioms are:

Level 2

VOC2

CON2: HOME

PRED2: carry, at, loc, room

ACT3: grab, drop, go

Gs2:

$\forall$ x y ((carry(x) $\wedge$ at(y)) $\supset$ loc(x,y))

$\forall$ x y z ((loc(x,y) $\wedge$ loc(x,z)) $\supset$ y=z)

Gd2:

go: main axiom

$\forall$ strt fin ((at(strt) $\wedge$ room(fin))
$\supset$ [go(strt,fin)] at(fin))

go: frame axioms

$\forall$ strt fin gfobj gfrm ((loc(gfobj,gfrm) $\wedge$ $\neg$carry(gfobj))
$\supset$ [go(strt,fin)] loc(gfobj,gfrm))

$\forall$ strt fin gfobj (carry(gfobj)
$\supset$ [go(strt,fin)] carry(gfobj))

$\forall$ strt fin gfobj ($\neg$carry(gfobj)
$\supset$ [go(strt,fin)] $\neg$carry(gfobj))

grab, drop: main axioms

$\forall$ obj rm ((at(rm) $\wedge$ loc(obj,rm))
$\supset$ [grab(obj)] carry(obj))

$\forall$ obj (carry(obj)

⊃ [drop(obj)] ¬carry(obj))

grab, drop: frame axioms

∀ obj fobj ((¬carry(fobj) ∧ fobj≠obj)
⊃ [grab(obj)] ¬carry(fobj))

∀ obj fobj ((carry(fobj) ∧ fobj≠obj)
⊃ [drop(boj)] carry(fobj))

∧ pure frame axioms:
grab: carry, loc, at
drop: ¬carry, loc, at

The level 2 actions are used to solve the level 1 domain constraints. But to synthesize a plan to direct the robot anywhere, his initial location must be known -- information which does not explicitly appear in the high-level description. One way around this is to simply insist that the robot is at some particular known location -- for instance, HOME -- in every state in the high-level plan. This is (a bit underhandedly) assured by translating the "lloc(x,y)" predicate which appears in the pre and post condition of every level 1 dynamic axiom as "x is in room y, and the robot is at HOME". So Gs2 is augmented by

∀ x (lloc(x) ≡ (at(HOME) ∧ loc(x)))

The planner has the choice of precomputing a level 2 solution to the constraint set G1(transfer(tobj,tstrt,tfin)), or merely solving the individual instances of transfer as they arise. The former course is preferable here. The level 2 planning problem

<VOC2, G2, G1(transfer(tobj,tstrt,tfin))>

contains two plan constraints; the initial call to BIGRESS is

BIGRESS(< lloc(tobj,tstrt) ∧ room(tfin) ,
lloc(tfobj,tfrm) ∧ tfobj≠tobj >,

< lloc(tobj,tfin) ,
lloc(tfobj,tfrm) >)

The main and frame axioms for transfer are treated as separate problems which must have the same solution. The variables tobj, tstrt, and tfin may appear in the solution, since they will be replaced by particular constants when an instance of transfer is executed. The variables tfobj and tfrm may not; there is no one object you can point to and say, "this is the object not being moved!" The chart below traces the parallel sequences of state descriptions developed on a forward search.

	r1	r2
initial	lloc(tobj,tstrt) ∧ room(tfin)	lloc(tfobj,tfrm) ∧ tfobj≠tobj
	≡ loc(tobj,tstrt) ∧ at(HOME) ∧ room(tfin)	≡ loc(tfobj,tfrm) ∧ at(HOME) ∧ tfobj≠tobj
/go(HOME,tstrt)	loc(tobj,tstrt) ∧ at(tstrt)	at(tstrt) ∧ ((carry(tfobj) ∧ tfrm=HOME) v loc(tfobj,tfrm))
/grab(tobj)	loc(tobj,tstrt) ∧ at(tstrt) ∧ carry(tobj)	"
/go(tstrt,tfin)	at(tfin) ∧ carry(tobj)	at(tfin) ∧ ((carry(tfobj) ∧ tfrm=HOME) v loc(tfobj,tfrm))
/drop(tobj)	at(tfin) ∧ ¬carry(tobj) ∧ loc(tobj,tfin)	"
/go(tfin,HOME)	at(HOME) ∧ ¬carry(tobj) ∧ loc(tobj,tfin)	at(HOME) ∧ ((carry(tfobj) ∧ tfrm=HOME) v loc(tfobj,tfrm))
	≡ lloc(tobj,tfin)	≡ lloc(tfobj,tfrm)

Correctness of BIGRESS lets us conclude that

$$G2 \vdash \forall\ tobj\ tstrt\ tfin\ ((lloc(tobj,tstrt) \wedge room(tfin)) \supset [go(HOME,tstrt);grab(tobj);go(tstrt,tfin);drop(tobj);go(tfin,HOME)]\ lloc(tobj,tfin))$$

$$G2 \vdash \forall\ tobj\ tstrt\ tfin\ tfobj\ tfrm\ ((lloc(tfobj,tfrm) \wedge tfobj \neq tobj) \supset [go(HOME,tstrt);grab(tobj);go(tstrt,tfin);drop(tobj);go(tfin,HOME)]\ lloc(tfobj,tfrm))$$

Specialization (as in the theorem stated earlier) gives low level plans to perform the particular go atomic actions (go(HOME,CAVE), etc.) needed in the top-level plan.

Grab and Drop are simply atomic actions at the third level as well, but go is replaced by the walk action, which takes a compass direction as its single parameter. Thus,

Level 3

VOC3

CON3: HOME, CAVE, ..., LAIR
N, E, S, W

PRED3: loc, at, carry

ACT3: walk, grab drop

Gs3: same as Gs2

Gd3:

grab, drop: same as Gd2

walk: frame axioms

∀ dir fobj frm ((loc(fobj,frm) ∧ ¬carry(fobj))
⊃[walk(dir)] loc(fobj,frm))

∀ dir fobj (carry(fobj)
⊃[walk(dir)] carry(fobj))

∀ dir fobj (¬carry(fobj)
⊃[walk(dir)] ¬carry(fobj))

walk: main axioms

at(HOME) ⊃[walk(N)] at(CAVE)
at(CAVE) ⊃[walk(S)] at(HOME)
at(CAVE) ⊃[walk(E)] at(LAIR)
at(LAIR) ⊃[walk(W)] at(CAVE)
•
•
•

It is not very feasible to precompute the solutions to go at this level; since a different plan is required to go between any two locations, a general solution would contain 2N**2 conditional branches, where N is the number of rooms.

The planning problem <VOC3, G3, G2(go(HOME,CAVE))> has four constraints -- one for the main go axiom, plus each frame axiom. Some of the level 3 solutions:

go(HOME,CAVE)	walk(N)
go(HOME,HOME)	null
go(HOME,LAIR)	walk(N);walk(E)

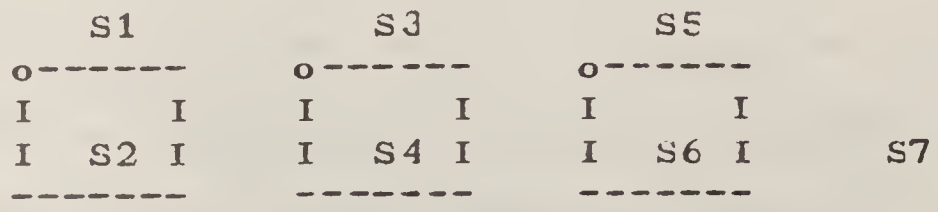
Thus the overall low-level solution to the example planning problem is

```
walk(N); grab(GOLD); walk(S); drop(GOLD); walk(N);
walk(E); grab(JEWELS); walk(W); walk(S); drop(JEWELS)
```

The hierarchy has considerably reduced the low-level search space, but at the cost of efficiency in the overall plan. Note that the low-level axiomatization allows the robot to carry more than one object. A better solution would be to walk to the CAVE, grab the GOLD, proceed directly to the LAIR, and return HOME. Alas, this is one of the classic tradeoffs: search efficiency versus optimality of the solution.

6.2 MORE BLOCKS AND BOXES

What is a "good" high level view of the block and boxes world, described in section 5.1? Recall that the world had three boxes, a table, and a number of blocks which could rest on a box or the table, or in a box.



There are seven surfaces on which a block can rest, labeled S1-S7 above. Any planning problem will probably involve simply rearranging the blocks on these surfaces. The high level world description contains a single action move(obj,olds,news), to move a

block from one surface to another.

Next, we must decide what side effects, if any, move has. Taking a block out of a box (from S2, S4, S6) disturbs any blocks resting on the box (on S1, S3, S5). "Hiding" this fact from the top level view would either lead to a "heuristic", rather than an exact, hierarchy of levels, or require that the low level plan which implements move explicitly undo the side-effects. While the latter course may be attractive in some cases, more efficient high-level plans (at the cost of some extra effort in calculating the regressions and progressions through move) are obtained by percolating the side effects all the way to the top.

Surfaces 1,3,5 are marked as "above" surfaces 2,4,6. The frame axioms for move check whether each block not being explicitly moved is above either the source or destination surfaces.

VOC1: BLK1, ..., BLK10
S1, ..., S7

PRED1: surface, above, ons

Gs1:

$\forall$ x (surface(x) $\equiv$ (x=S1 $\vee$ ... $\vee$ x=S7))

$\forall$ x y (above(x,y) $\equiv$ (
(x=S1 $\wedge$ y=S2) $\vee$ ... $\vee$ (x=S5 $\wedge$ y=S6)))

$\forall$ x y (ons(x,y) $\supset$ surface(y))

Gd1:

$\forall$ obj olds news ((ons(obj,olds) $\wedge$ surface(news))
$\supset$[move(obj,olds,news)] ons(obj,news))

$\forall$ obj olds news w z
((ons(w,z) $\wedge$ w$\neq$obj $\wedge$ $\neg$above(z,olds) $\wedge$ $\neg$above(z,news))
$\supset$[move(obj,olds,news)] ons(w,z))

$\forall$ obj olds news w z
((ons(w,z) $\wedge$ (above(z,olds) $\vee$ above(z,news)))
$\supset$[move(obj,olds,news)] ons(obj,S7))

Given these axioms, the problem discussed in 5.1 to move two blocks from inside BOX1 to its top uses the constraint

$$(ons(BLK1,S2) \wedge ons(BLK2,S2)) \supset[u] (ons(BLK1,S1) \wedge ons(BLK2,S1))$$

Backwards search by BIGRESS finds a solution, probably without any backtracking. LiveBackward(ons(BLK1,S1) ∧ ons(BLK2,S1)) will not contain move(BLK1,S2,S1) or move(BLK2,S2,S1); because of the constraints on the frame axioms, the regression of ons(BLK1,S1) ∧ ons(BLK2,S1) through either action is false, and thus pruned. So BIGRESS must choose a different spot from which to obtain a block; move(BLK1,S7,S1) will do. The regressed goal becomes ons(BLK1,S7) ∧ ons(BLK2,S1). Next, the choice of move(BLK1,S1,S7) is pruned; the only reasonable choice is move(BLK2,S2,S1) followed (finally) by move(BK1,S2,S7). So the high level solution is

move(BLK1,S2,S7);
move(BLK2,S2,S1);
move(BLK1,S7,S1)

The low level vocabulary and axioms are given in 5.1, augmented by translation axioms

$$\forall x\ (ons(x,S1) \equiv (on(x,BOX1) \wedge handempty))$$
$$\forall x\ (ons(x,S2) \equiv (in(x,BOX1) \wedge handempty))$$
$$\forall x\ (ons(x,S3) \equiv (on(x,BOX2) \wedge handempty))$$
$$\forall x\ (ons(x,S4) \equiv (in(x,BOX2) \wedge handempty))$$
$$\forall x\ (ons(x,S5) \equiv (on(x,BOX3) \wedge handempty))$$
$$\forall x\ (ons(x,S6) \equiv (in(x,BOX3) \wedge handempty))$$
$$\forall x\ (ons(x,S7) \equiv (on(x,TABLE) \wedge handempty))$$

The "closed" predicate does not appear in the high-level vocabulary or in the translation axioms; the solutions to move use a test to determine this detail. The results of the low level planning are

high level	implemented by
move(BLK1,S2,S7)	(closed(BOX1) → open(BOX1),null); pickup(BLK1,BOX1); putdown(BLK1,TABLE)
move(BLK2,S2,S1)	(closed(BOX1) → open(BOX1),null); pickup(BLK2,BOX1); close(BOX1); putdown(BLK2,BOX1)
move(BLK1,S7,S1)	(closed(BOX1) → null,shut(BOX1)); pickup(BLK1,S7); putdown(BLK1,S1)

The final solution is just, of course, the concatentation of the three plans above. The second and third test of closed(BOX1) are not necessary (they always fail). Perhaps a post-processing stage could optimize the plan by removing unneeded tests.

A heuristic hierarchical planner would most likely suppress the side-effect of move as a detail, and initially come up with the "buggy" high level plan [move(BLK1,S2,S1);move(BLK2,S2,S1)]. While the cost of repairing this particular plan may be cheap, it is interesting to note that retention of the side-effect in the exact decompostion actually helped constrain the high-level search.

6.3 SUMMARY

A hierarchical decomposition of a planning problem can greatly increase search efficiency. Imposing a hierarchy involves not only decisions as to the tools available at each level, but also which subproblems should be (can be) solved in advance. The use of plan parameters permits the synthesis of general solutions to planning problems.

A strict hierarchy may force more detail to the top level than a heuristic hierarchy, but it is not clear if such detail necessary

degrades search efficiency. (Of course, even an exact hierarchy rarely retains all the detail that a MACROP does.)

The strict hierarchy effectively isolates each sister node, eliminating the need to check for bad interactions. But it may be very desirable to allow some "communication" between sisters. For example, when solving the second subproblem in the last example, the planner should have known that BOX1 was left open at the end of the first subtask. In other words, an ordered tree may be easier to solve than an unordered tree. The exact formal mechanism for carrying information from one sister to the next remains to be defined.

CHAPTER 7

FORMAL OBJECTS

7.1 BACKGROUND

The amount of search performed by a problem solver can often be drastically reduced by neuristically ordering the successors to each node in the search graph. But it may be difficult to discern the "best" successor until later in the planning process; or a number of successors may lead to essentially the same solution or dead-end. In such circumstances it is advantageous to apply a partially specified operator to the current node, only completing the specification at a later point in the search. This effectively merges a number of descendent branches, while retaining the option of splitting off branches as necessary.

Formal objects are used in one version of this technique [Sussman 73]. A formal object is a "dummy" term introduced by the planner; initially, at least, the formal object's referent is only partially specified. A simple action containing a formal object as a parameter can thus stand for a range of constant-parameter actions.

Mechanisms for manipulating formal objects, or "uninstantiated parameters", have appeared in a number of linear and non-linear planning systems, including ABSTRIPS and NOAH. We give a high-level description of the use of formal objects in the TDL framework. This work is valuable in providing a very general basis for the use of formal objects in both forward and backward search, and at least a semi-formal treatment of the correctness of plans containing formal objects.

7.2 BIGRESS WITH FORMAL OBJECTS

The formal objects AllFormObjs are a distinct subset of the variables.

A restriction wff is a quantifier-free non-modal formula containing only invariant predicates.

Rwffs(F), where F is a list of variables, is the set of restriction wffs whose free variables are contained in (F u H). (H, we recall, is the list of plan parameters.)

BIGRESS takes as parameters:

BIGRESS(R, S, FormObjs, Restrict)

where FormObjs $\subset$ AllFormObjs

Restrict ϵ Rwffs(FormObjs)

and returns a triple

<Plan, PlanFormObjs, PlanRestrict>

such that:

(i) H u PlanFormObjs contains all variables free in Plan

(ii) FormObjs ⊆ PlanFormCbjs ⊂ AllFormObjs

(iii) G⊢ PlanRestrict ⊃ (ri ⊃ [Plan] si)
for 1<=i<=k

(iv) G⊢ PlanRestrict ⊃ Restrict

(v) G⊢ (E PlanFormObjs) PlanRestrict
and treating this test as a query,
an answer exists in which each variable
in PlanFormObjs is bound to a single
constant, plan parameter, or ∀-quantified
variable. This condition shall be
abbreviated as "⊢*".

Thus PlanRestrict specifies the sets of bindings for the formal objets which make the Plan a solution to the planning problem. Condition (v) insures that the planner knows of at least one suitable set of bindings.

SOLVE(R,S) := BIGRESS(R,S, {}, true)

```
BIGRESS(R, S, FormObjs, Restrict):

   If G⊢ ri⊃si for 1<=i<=k then
           return <null, FormObjs, Restrict>.

   Choose:

      Choose <A, R/A> from LiveForward(R, FormObjs);

         <Subplan, NewFormObjs, NewRestrict> :=
            BIGRESS(R/A, S, FormObjs, Restrict);

         return <A;Subplan, NewFormObjs, NewRestrict>.

      Choose <A, R\A> from LiveBackward(S, FormObjs);

         <Subplan, NewFormObjs, NewRestrict> :=
            BIGRESS(R, A\S, FormObjs, Restrict);

         return <Subplan;A, NewFormObjs, NewRestrict>.

      Choose <NewFormObjs, NewRestrict> from
                                       Merge(FormObjs);

         return BIGRESS(R ∧ NewRestrict, S,
                        FormObjs u NewFormObjs,
                        Restrict ∧ NewRestrict).

      Choose NewRestrict from Diverge(FormObjs, Restrict);

         return BIGRESS(R ∧ NewRestrict,
                        S ∧ NewRestrict,
                        FormObjs, NewRestrict).

      Choose TEST from NonTriv(R, FormObjs);

         <Sub1, NewForm1, NewRes1>:=
            BIGRESS(R ∧ TEST, S, FormObjs, Restrict);

         <Sub2, NewForm2, NewRes2>:=
            BIGRESS(R ∧ ¬TEST ∧ NewRes1, S, NewForm1, NewRes1);

         return <(TEST → Sub1, Sub2), NewForm2, NewRes2).

end BIGRESS


LiveForward, LiveBackward, NonTriv:

   Same as before, except that members of
   FormObjs may appear free in A or W.
```

```
Merge(FormObjs):

   return {<NewFormObjs, NewRestrict> |
      NewFormObjs ⊂ AllFormObjs - FormObjs
      NewRestrict ∈ Rwffs(NewFormObjs ∪ FormObjs)
      G⊢* (E NewFormObjs) NewRestrict}.


Diverge(FormObjs, Restrict):

   return {NewRestrict |
      NewRestrict ∈ Rwffs(FormObjs)
      G⊢ NewRestrict ⊃ Restrict
      not G⊢ Restrict ⊃ NewRestrict
      G⊢* (E FormObjs) NewRestrict
      (not G⊢ ¬(Restrict ∧ si))
         for 1<=i<=k}
```

Merge introduces new formal objects. It selects an Rwff NewRestrict which constrains the values of the new objects only. These formal objects are then available to LiveForward, Livebackward, or NonTriv. For example, Merge may return <f, f=C1 v f=C2>; BIGRESS can then choose <a(f), R/a(f)> from Liveforward, merging the paths through a(C1) and a(C2).

Diverge increases the constraint on the possible values of the currently available formal objects. This permits simplification of world descriptions and/or success of the stopping test. For example, if Restrict is f=C1 v f=C2, and the portion of the plan already found is a(f);b(f), choosing a NewRestrict of F=C1 splits off the path through a(C1);b(C1). The final condition in Diverge is a heuristic that avoids constraining the formal objects in such a way that the goal states become unobtainable.

The modifications to the introduction of conditionals by BIGRESS are necessary to insure that both branches receive

compatible bindings for previously introduced formal objects.

7.3 CORRECTNESS

It is merely necessary to show that conditions (i) - (v) above hold.

(i) and (ii) are obviously true. PlanRestrict is simply the final parameter to BIGRESS on its last recursive call. Diverge explicitly maintains conditions (iv) and (v), and it is readily seen that Merge does as well:

(Restrict ∧ NewRestrict) ⊃ Restrict

is a tautology, and if

G⊢* (E NewFormObjs) NewRestrict

and

G⊢* (E FormObjs) Restrict

then certainly

G⊢* (E NewFormObjs)(E FormObjs)

(Restrict ∧ NewRestrict)

Condition (iii) is proved by induction on N, the number of calls to BIGRESS. Attach the phrase "for 1<=i<=k" after each formula involving ri and si in the following sketch.

If N=1, then plan=null and G⊢ ri⊃si. So for any formula PlanRestrict, G⊢ PlanRestrict⊃(ri⊃[null]si).

So suppose (iii) holds for any number of calls less than N. The top level of BIGRESS chooses one of:

--LiveForward: The recursive call constructs a Subplan such that

G⊢ PlanRestrict ⊃ (ri/A ⊃ [Subplan]si)

So

G⊢ [A]PlanRestrict ⊃ (ri ⊃ [A;Subplan]si)

The reason for using only invariant predicates in restriction wffs becomes clear: it means that

G⊢ PlanRestrict ⊃ [A] PlanRestrict

so that we obtain the desired condition,

G⊢ PlanRestrict ⊃ (ri ⊃ [A;Subplan]si)

An interesting generalization may be to allow non-invariant predicates in restriction wfis, and explicitly progress or regress the restriction through the action A.

--LiveBackward, NonTriv: Are handled similarly.

--Merge: By the induction hypothesis,

G⊢ PlanRestrict ⊃ (ri ∧ NewRestrict ⊃[Plan]si)

So

G⊢ (PlanRestrict ∧ NewRestrict) ⊃ (ri ⊃[Plan]si)

By (iv),

G⊢ PlanRestrict ⊃ Restrict ∧ NewRestrict

Propositional reasoning allows us to derive

G⊢ PlanRestrict ⊃ (ri ⊃[Plan]si)

--Diverge: Similar to Merge.

7.4 INTELLIGENCE

Some intelligence needs to be built into Merge and Diverge before they can add any "power" to BIGRESS. It is useful to introduce formal objects when the current LiveForward or LiveBackward sets contain a large number of actions which differ only in one or two parameters. Heuristics could be included, for example, to introduce a formal object whose new restriction wff is a disjunction of equalities with each of these parameters, or is the proposition that the formal object is of the proper type to be a parameter of the action.

The new restriction wff can simply be true for formal objects used in backwards search. The possible values for the formal object are "automatically" constrained by the regression process. For example, suppose the sole dynamic axiom is:

$$\forall xy(p1(x) \wedge p2(y) \supset [a(x,y)]\, q(x))$$

and s=q(C). Then Merge can introduce a formal object f with NewRestrict=true, so that Restrict is unchanged. Now $a(C,f)\backslash q(C) = p1(C) \wedge p2(f)$. In order to reach this new goal state, p2(f) must be be achieved.

As we saw in earlier examples, world descriptions become quite unwieldy when large numbers of free variables are present. Diverge can be selected in such cases to simplify R and S. When a formal object f is constrained to be equal to a single constant C, one can replace all instances of f by C in R and S, and systematically evaluate the equality predicates in R and S which involve C.

BIGRESS should know when a stronger restriction wff can lead immediately to a solution to the planning problem. Each positive answer to the query

$G \vdash^* (E\ FormObjs)(Restrict \wedge (Restrict \supset (ri \supset si)))$

which holds across all the constraints can be used to construct such a restriction wff. This special case of Diverge could be efficiently incorporated in the stopping test itself.

7.5 EXAMPLE

Use the level 1 dungeons world axioms. Let Res be

$loc(GOLD,LAIR) \supset [u]\ loc(GOLD,CAVE)$

In

BIGRESS(loc(GOLD,LAIR), loc(GOLD,CAVE), {}, true)

the set

LiveForward(loc(GOLD,CAVE))

contains transfer(GOLD,LAIR,x) for every room x. Instead of searchinng through transfer(GOLD,LAIR,HOME), transfer(GOLD,LAIR,PIT), etc. (until finally hitting on transfer(GOLD,LAIR,CAVE)) BIGRESS can choose

<{f}, room(f)>

from Merge({}), and recursively call

BIGRESS(loc(GOLD,LAIR) ∧ room(f), loc(GOLD,CAVE)).

Then

<transfer(GOLD,LAIR,f), loc(GOLD,f) ∧ room(f)>

should be chosen from

LiveForward(loc(GOLD,LAIR) ∧ room(f)),

effectively merging all forward branches. The stopping test in

BIGRESS(loc(GOLD,f) ∧ room(f), loc(GCLD,CAVE)) would be satisfied if f=CAVE; so BIGRESS should choose f=CAVE from

Diverge({f}, room(f)).

The innermost call

BIGRESS(loc(GOLD,f) ∧ room(f) ∧ f=CAVE, loc(GOLD,CAVE), {f}, room(F) ∧ f=CAVE)

terminates the search, returning

<transfer(GOLD,LAIR,f), {f}, room(f) ∧ f=CAVE>

7.6 SELECT STATEMENT

The final restriction wff may be more general than a conjunction of equalities. The plan constraint

loc(SWORD,CAVE) =[u] ¬loc(SWORD,CAVE)

could lead to the solution set

<transfer(SWORD,CAVE,f>, {f}, room(f) ∧ f≠CAVE>

How shall the planner make use of such a solution? It could immediately find a constant binding for f, and substitute it into the plan. But if this plan is to be incorporated into a larger sequence of actions, it may be more efficient in the long run to choose the bindings for formal objects at execution time. This can be done by inserting a new type of statement, select, at the beginning of the plan.

A select statement is written

SEL(f1,...,fn)Res

where the fi's are variables and Res is a nonmodal wff. We shall only admit a select as the first statement of a solution.

A solution set of the form

<Plan, PlanFormObjs, PlanRestrict>

is transformable to the plan

SEL(PlanFormObjs)PlanRestrict;Plan

The solution to the example above can thus be written

SEL(f) (room(f) ∧ f≠CAVE); transfer(SWORD,CAVE,f)

and we'd expect that

loc(SWORD,CAVE) ⊃[SEL(f) (room(f) ∧ f≠CAVE);
transfer(SWORD,CAVE,f)]
¬loc(SWORD,CAVE)

SEL can be given precise semantics and axiomatics:

M(SEL(f1,...,fn)Res) =
{(I,J) | E d1,...,dn ∈ DOMAIN such that
{d1/f1,...,dn/fn}I = J
J|= Res}

[SEL(f1,...,fn)Res]Q ≡ (∀ f1,...,fn)(Res ⊃ Q)

It is then easy to verify that if no formal objects appear in R (as on the first call to BIGRESS), that

G⊢ Res ⊃ (R ⊃ [u] S)

iff

G⊢ R ⊃ [SEL(FormObjs)Res;u] S

Select <u>preserves termination</u> iff

G⊢ <SEL(FormObjs)Res> true

or equivalently

G⊢ (E FormObjs) Res

--which is a condition for Res to be an acceptable restriction wff.

Select is the only <u>non-transparent</u> action we've discussed, and as such really falls beyond the scope of the current paper. A more complete development of the select action would lead into general

assignment actions.

CHAPTER 8

CONCLUSIONS

8.1 SUMMARY

This thesis examined planning formalisms and systems based on the situational calculus, state-space models, and hierarchical problem decomposition, and their influence on Rosenschein's formulation of planning in propositional dynamic logic.

I extended the formulation to Transparent Dynamic Logic, a special version of first-order dynamic logic which includes parameterized actions and allows substitution of terms through modal contexts. Rosenschein's BIGRESS algorithm was extended for a subset of the language and proved correct. Goals and state descriptions could contain both quantified and disjunctive information.

A number of examples of single-level and hierarchical planning problems were worked out which demonstrated the strengths of the dynamic logic approach over STRIPS, non-linear planners, and those with non-exact plan hierarchies. Finally, an approach to the use of formal objects in the plan generation process was sketched.

8.2 FUTURE WORK

The most gaping holes in the present work concern completeness of the progression and regression functions. We need a syntactic characterization of the first-order planning problems for which the answer-generation process can be guaranteed to find all significant answers and always terminate. Next, we need a proof (probably semantic) that the formula derived from all the answers is, in fact, the strongest provable postcondition (weakest provable precondition).

Many extensions to the framework suggest themselves:

It is desirable to be able to refer to entities for which no constant or plan parameter is known. The present system cannot handle a command like, "pick up the block on the table". The Select action, discussed in Chapter 7, provides a mechanism for explicitly binding a variable to an entity which satisfies some description at a specified point in the plan.

We made actions transparent by effectively banning non-rigid terms other than variables. It would be useful in the Dungeons World to have a term "location-of-robot" whose referent is changed by the "go" action. But if any action is allowed to change the referent of any term, we could be stuck with frame axioms for the unchanged terms.

Dynamic logic includes a transitive closure operator "*", not described in this thesis, which can be used with the ? and U operators to construct loops. We'd like to extend BIGRESS to

construct plans with loops, such as "Put all the blocks in BOX1", or "Walk north until you reach CASTLE9".

8.3 CONCLUDING REMARKS

No attempt has been made to implement the system described in this thesis.* A practical implementation must carefully control the amount of theorem proving performed in the stopping test and progression and regression functions. Although frame axioms are given a uniform formal treatment with other dynamic axioms, the theorem prover should limit their role in the resolution process, as suggested in section 4.5.6.. It remains very costly to handle equality in full generality; a preliminary implementation may prohibit the use of existentially-quantified variables and plan parameters, thus eliminating the need for frame axioms for equality.

Still, the dynamic logic framework is useful if only for its ability to describe and formalize various approaches to the planning problem. The world does not need another program that solves the three-block problem; an elegant, incrementally extendable theory of planning is of considerable interest.

---

*An implementation based on an extension of the propositional framework to propositional schemata has been completed at SRI; details are expected to appear in [Rosenschein (forthcoming)].

Bibliography

Ackermann, Wilhelm (1954) Solvable Cases of the Decision Problem.

Chang, C.L. and Lee, R.C.T. (1973) Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York.

Cohen, Phillip R. (1978) On Knowing What to Say: Planning Speech Acts. TR 118, University of Toronto, Dept. of Computer Science.

Dawson, C., and Siklossy, L. (1977) The role of preprocessing in problem solving systems. Proceedings of the 5th International Joint Conference on Artifical Intelligence, Massachusetts Institute of Technology, Cambridge, MA.

Fahlman, S.E. (1974) A Planning System for Robot Construction Tasks. Artifical Intelligence, 5(1), 1-49.

Fikes, R.E. (1975) Deductive retrieval mechanisms for state description models. Advance Papers of the Fourth International Joint Conference on Artifical Intelligence. Tbilisi, Georgia, USSR.

Fikes, R.E. and Nilsson, N.J. (1971) STRIPS: a new approach to the application of theorem proving to problem solving. Artifical Intelligence, 2(3/4), 189-208.

Green, C. (1969a) Application of theorem proving to problem solving. International Joint Conference on Artifical Intelligence, (D. Walker and L. Norton, Eds.) Washington, D.C.

Green, C. (1969b) Theorem-proving by resolution as a basis for question-answering systems. Machine Intelligence 4, Edinburgh University Press, Edinburgh.

Haas, Andrew (1981) Naive Psychology for Planning, University of Rochester, Dept. of Computer Science (PhD thesis).

Harel, David (1979) First Order Dynamic Logic, Springer-Verlang.

Lewis, H.R. (1975) Cycles of Unifiability and Decidability by Resolution. Tech Report, Aiken Computation Laboratory, Harvard University Cambridge, MA.

McCarthy, J. and Hayes, P. (1969) Some philosophical questions from the standpoint of artifical intelligence. Machine Intelligence 4, Edinburgh University Press, Edinburgh.

Moore, Robert (1980) Reasoning About Knowledge and Action, SRI Tech Note 191.

Newell, A., Shaw, J.C., and Simon, H.A. (1960) Report on a general problem-solving progra for a computer. Information Processing: Proceedings of the International Conference on Information Processing, UNESCO, Paris.

Nilsson, Nils J. (1980) Principles of Artifical Intelligence, Tiago Publishing Co., Palo Alto, CA.

Pratt, Vaughan R. (1976) Semantical Considerations of the Floyd-Hoare Logic. Proceedings of the 17th IEEE Symposium on the Foundations of Computer Science.

Reiter, R. (1978) On Structuring a First Order Data Base. Proceedings of the Canadian Society for Computational Studies of Intelligence (R. Perrault, Ed.) Univerisity of Toronto.

Robinson, G.A. and Wos, L. (1969) Paramodulation and Theorem Proving in First Order Theories with Equality. Machine Intelligence 4, (B. Meltzer and D. Michie, Eds.) American Elsevier, New York.

Rosenschein, Stanley J. (1981) Plan synthesis: a logical approach. Proceedings of the 8th International Joint Conference on Artifical Intelligence, University of British Columbia, Vancouver, BC.

Rosenschein, Stanley J. (forthcoming) Hierarchical Planning: Implementation Considerations. SRI Technical Report.

Sacerdoti, E. (1974) Planning in a hierarchy of abstraction spaces. Artifical Intelligence, 5(2), 115-135.

Sacerdoti, E. (1974) A Structure for Plans and behavior, American Elsevier, New York.

Sussman, G.J. (1977) A Computer Model of Skill Acquisition, American Elsevier, New York.

Waldinger, R.J. (1977) Achieving several goals simultaneously. Machine Intelligence 8, Ellis Horwood, Chichester.

University of Toronto
Computer Systems Research Group

BIBLIOGRAPHY OF CSRG TECHNICAL REPORTS 1980 - present

* - Out of print

* CSRG-108 DIALOGUE ORGANIZATION AND STRUCTURE FOR INTERACTIVE INFORMATION SYSTEMS
John Leonard Barron
[M.Sc. Thesis, DCS, 1980]

* CSRG-109 A UNIFYING MODEL OF PHYSICAL DATABASES
D.S. Batory, C.C. Gotlieb, April 1980

* CSRG-110 OPTIMAL FILE DESIGNS AND REORGANIZATION POINTS
D.S. Batory, April 1980

* CSRG-111 A PANACHE OF DBMS IDEAS III
D. Tsichritzis (ed.), April 1980

CSRG-112 TOPICS IN PSN - II: EXCEPTIONAL CONDITION HANDLING IN PSN; REPRESENTING PROGRAMS IN PSN; CONTENTS IN PSN
Yves Lesperance, Byran M. Kramer, Peter F. Schneider
April, 1980

CSRG-113 SYSTEM-ORIENTED MACRO-SCHEDULING
C.C. Gotlieb and A. Schonbach
May 1980

CSRG-114 A FRAMEWORK FOR VISUAL MOTION UNDERSTANDING
John Konstantine Tsotsos
[Ph.D. Thesis, DCS, June 1980]

CSRG-115 SPECIFICATION OF CONCURRENT EUCLID
James R. Cordy and Richard C. Holt
July 1980

CSRG-116 THE REPRESENTATION OF PROGRAMS IN THE PROCEDURAL SEMANTIC NETWORK FORMALISM
Bryan M. Kramer
[M.Sc. Thesis, DCS, 1980]

CSRG-117 CONTEXT-FREE GRAMMARS AND DERIVATION TREES AS PROGRAMMING TOOLS
Volker Linnemann
September 1980

CSRG-118 S/SL: SYNTAX/SEMANTIC LANGUAGE INTRODUCTION AND SPECIFICATION
R.C. Holt, J.R. Cordy, D.B. Wortman
CSRG, September 1980

CSRG-119 PT: A PASCAL SUBSET
Alan Rosselet
[M.Sc. Thesis, DCS, October 1980]

CSRG-120 PTED: A STANDARD PASCAL TEXT EDITOR BASED ON THE KERNIGHAN AND PLAUGER DESIGN
Ken Newman, DCS
October 1980

CSRG-121 TERMINAL CONTEXT GRAMMARS
Howard W. Trickey
[M.Sc. Thesis, EE, September 1980]

CSRG-122 THE APPROXIMATE SOLUTION OF LARGE QUEUEING NETWORK MODELS
John Zahorjan
[Ph.D. Thesis, DCS, August 1980]

CSRG-123 A FORMAL TREATMENT OF IMPERFECT INFORMATION IN DATABASE MANAGEMENT
Yannis Vassiliou
[Ph.D. Thesis, DCS, September 1980]

CSRG-124 AN ANALYTIC MODEL OF PHYSICAL DATABASES
Don S. Batory
[Ph.D. Thesis, DCS, January 1981]

CSRG-125 MACHINE-INDEPENDENT CODE GENERATION
Richard H. Kozlak
[M.Sc. Thesis, DCS, January 1981]

CSRG-126 COMPUTER MACRO-SCHEDULING FOR HIGH PRODUCTIVITY
Abraham Schonbach
[Ph.D. Thesis, DCS, March 1981]

CSRG-127 OMEGA ALPHA
D. Tsichritzis (ed.), March 1981

CSRG-128 DIALOGUE AND PROCESS DESIGN FOR INTERACTIVE INFORMATION SYSTEMS USING TAXIS
John Barron, April 1981

CSRG-129 DESIGN AND VERIFICATION OF INTERACTIVE INFORMATION SYSTEMS USING TAXIS
Harry K.T. Wong
[Ph.D. Thesis, DCS, to be submitted]

CSRG-130 DYNAMIC PROTECTION OF OBJECTS IN A COMPUTER UTILITY
Leslie H. Goldsmith, April, 1981

CSRG-131 INTEGRITY ANALYSIS: A METHODOLOGY FOR EDP AUDIT AND DATA QUALITY CONTROL
Maija Irene Svanks
[Ph.D. Thesis, DCS, February 1981]

CSRG-132 A PROTOTYPE KNOWLEDGE-BASED SYSTEM
FOR COMPUTER-ASSISTED MEDICAL DIAGNOSIS
Stephen A. Ho-Tai
[M.Sc.Thesis, DCS, January 1981]

CSRG-133 SPECIFICATION OF CONCURRENT EUCLID
James R. Cordy, Richard C. Holt
August 1981 (Version 1)

CSRG-134 ANOTHER LOOK AT COMMUNICATING PROCESSES
E.C.R. Hehner and C.A.R. Hoare, July, 1981

CSRG-135 ROBUST CONCURRENCY CONTROL IN DISTRIBUTED DATABASES
Derek L. Eager
[M.Sc. Thesis, DCS, October 1981]

CSRG-136 ESTIMATING SELECTIVITIES IN DATA BASES
Stavros Christodoulakis
[Ph.D. Thesis, DCS, December 1981]

CSRG-137 SATISFYING DATABASE STATES
Marc H. Graham
[Ph.D. Thesis, DCS, December 1981]

CSRG-138 IMPROVING THE PERFORMANCE OF DATA BASE SYSTEMS
Geovane Cayres Magalhaes
[Ph.D. Thesis, DCS, December 1981]

CSRG-139 A FORMAL TREATMENT OF INCOMPLETE KNOWLEDGE BASES
Hector J. Levesque
[Ph.D. Thesis, DCS, February 1982]

CSRG-140 AN OVERVIEW OF TUNIS: A UNIX LOOK-ALIKE
WRITTEN IN CONCURRENT EUCLID
R.C. Holt, February 1982

CSRG-141 ON PROVING THE ABSENCE OF EXECUTION ERRORS
W. David Elliott
[Ph.D. Thesis, DCS, September 1980]

CSRG-142 A METHODOLOGY FOR PROGRAMMING WITH CONCURRENCY
Christian Lengauer
[Ph.D. Thesis, DCS, April 1982]

CSRG-143 ALPHA BETA
F. Lochovsky (ed.), May 1982

CSRG-144 A FIRST-ORDER DYNAMIC LOGIC FOR PLANNING
Henry A. Kautz
[M.Sc. Thesis, DCS, May 1982]